Un Esquema de Programación Lógico Funcional Concurrente con Restricciones

TRABAJO FIN DE MÁSTER

en Programación y Tecnología Software

Marcos Miguel García Toledo

Directores:

Rafael Caballero Roldán
Rafael del Vado Vírseda (Colaborador externo)

Máster en Investigación en Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso 2010-2011[1]

[1]Trabajo presentado en la convocatoria de Septiembre de 2011 obteniendo la calificación de Sobresaliente.

Un Esquema de Programación Lógico Funcional Concurrente con Restricciones

Memoria correspondiente al
Trabajo de Fin de Máster presentada por

Marcos Miguel García Toledo

Dirigida por los profesores

Rafael Caballero Roldán y Rafael del Vado Vírseda

Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Septiembre de 2011

Autorización de Difusión

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: **Un Esquema de Programación Lógico Funcional Concurrente con Restricciones**, realizado durante el curso académico 2010-2011 bajo la dirección de Dr. Rafael Caballero Roldán y Dr. Rafael del Vado Vírseda en el Departamento de Sistemas Informáticos y Computación, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Marcos Miguel García Toledo
Septiembre de 2011

Abstract

In this work we sketch a concurrent extension for constraint functional logic programming languages which allows to model concrete declarative applications in which concurrency and constraint solving are necessary. We encode goal solving processes into the $CFLP(\mathcal{D})$ scheme, a uniform foundation for the semantics of functional and constraint logic programs over a parametrically given constraint domain $\mathcal{D}$, in order to provide a declarative model for the possibility of specify, implement and execute, at a higher level of abstraction, concurrent real-world applications, i.e., declarative constraint programs describing several processes which may be executed concurrently and cooperate together to achieve their specific tasks via demand-driven narrowing with definitional trees, declarative residuation, cooperative constraint solving, and communication or sharing critical resources by means of the synchronization on logic variables.

Keywords: Multiparadigm programming, constraints, concurrency.

Resumen en castellano

En este trabajo vamos a presentar una extensión concurrente para lenguajes de programación lógico funcionales con restricciones que permita modelar aplicaciones declarativas concretas donde la concurrencia y la resolución de restricciones sean necesarias. Codificaremos procesos dentro del esquema $CFLP(\mathcal{D})$, cuyos fundamentos teóricos sobre las semánticas de programas funcionales y lógicos con restricciones sobre un dominio de restricciones paramétrico $\mathcal{D}$, proporcionan un modelo declarativo que permite especificar, implementar y ejecutar, a un nivel superior de abstracción, aplicaciones concurrentes reales, es decir, programas declarativos con restricciones que describen varios procesos para la resolución de objetivos que deben ser ejecutados concurrentemente y cooperar para llevar a cabo tareas específicas mediante estrechamiento dirigido por demanda con árboles definicionales, residuación declarativa, resolución cooperativa de restricciones, y comunicación o compartición de recursos críticos por medio de sincronización haciendo uso de variables lógicas.

Palabras clave: Programación multiparadigma, restricciones, concurrencia.

Notas acerca de este trabajo

Este trabajo ha sido realizado como una Tésis de Máster durante el curso académico 2010-2011 dentro del contexto del programa de Máster en Investigación en Informática de la Universidad Complutense de Madrid en la especialidad de Programación y Tecnología Software correspondiente al Departamento de Sistemas Informáticos y Computación. Este trabajo ha sido supervisado por Dr. Rafael Caballero Roldán y dirigido por Dr. Rafael del Vado Vírseda en el Grupo de Programación Declarativa (GPD). Por último, este trabajo ha sido financiado por los proyectos STAMP (TIN2008-06622-C03-01), Prometidos-CM (S2009TIC-1465), y GPD (UCM-BSCH-GR35/10-A-910502).

Agradecimientos

Me gustaría agradecer a todas las personas que han hecho posible la realización de este trabajo fin de Máster:

- A mi director Rafael Caballero Roldán, por haberme permitido continuar con este trabajo de fin de Máster junto a Rafael del Vado.
- A mi director Rafael del Vado Vírseda, por su investigación previa, por haber confiado en mí para este trabajo, por ayudarme siempre en todo lo posible e incluso más y por guiarme a lo largo de este trabajo.
- A mi familia, que siempre ha estado apoyándome y animándome, no solo a hacer las cosas mejor, sino a ser yo mismo mejor persona.

Contenido

Lista de Figuras

Capítulo 1

Introducción

Los esfuerzos hechos por extender la *programación declarativa* a lenguajes capaces de soportar sistemas concurrentes ha estimulado una investigación intensiva en las últimas décadas, dando como resultado una gran variedad de propuestas [7, 35], con la intención de desarrollar lenguajes de programación y sistemas declarativos concurrentes que mantengan un equilibrio deseable entre la expresividad y la lectura declarativa de los programas: abstracción, computaciones como demostraciones, facilidad para la metaprogramación, etc.

Una característica común de varias de estas propuestas es el intento de definir modelos declarativos con concurrencia dentro del paradigma de la *programación lógica* [8, 15, 45, 47, 75], un paradigma declarativo que permite resolver problemas de búsqueda combinatoria de estados gracias a su forma relacional y al *indeterminismo*. En el modelo declarativo clásico de interpretación de procesos [72, 78], un cómputo dentro de la programación lógica puede ser visto como la evolución de una red dinámica de procesos paralelos. Un átomo en el objetivo correspondería a un proceso y una variable que aparece en dos o más átomos representa un canal de comunicación entre los procesos correspondientes. De este modo, un objetivo representa una red de procesos paralelos conectados por canales de comunicación. La comunicación se modela por medio de la instanciación de variables durante la *unificación* de las mismas. Un paso de resolución da lugar a una reconfiguración de la red por expansión de un proceso (el átomo seleccionado) en una subred de procesos paralelos (los átomos se sustituyen por el cuerpo de las cláusulas seleccionadas), y estableciendo uniones con los canales de comunicación.

La *programación lógica con restricciones* (*CLP*) [9] generaliza el modelo computacional básico de la programación lógica, es decir, la unificación, sustituyendola por la *resolución de restricciones* sobre algún *dominio de restricciones* (por ejemplo, los enteros o los reales) definiendo un conjunto de restricciones primitivas y aportando

un *resolutor* de restricciones sobre el dominio. Un programa *CLP* sigue siendo un conjunto de cláusulas, pero ahora cada cláusula contendrá restricciones en su cuerpo. Por *restricción* entendemos un subconjunto (posiblemente infinito) del espacio de todas las posibles valoraciones que pueden tomar las variables. Para cada paso de derivación, en lugar de unificar dos átomos, el cómputo comprueba la satisfactibilidad de un conjunto de restricciones. *CLP* no es un único lenguaje sino que se define como una clase de lenguajes, el llamado *esquema* $CLP(\mathcal{D})$ [40] parametrizado por un dominio de restricciones $\mathcal{D}$. Una instancia de esta clase puede obtenerse considerando un dominio específico de restricciones y aportando un resolutor de restricciones para dicho dominio. El esquema $CLP(\mathcal{D})$ ha sido generalizado en el marco de la *programación con restricciones concurrente* [70, 71] para integrar las ideas de la programación lógica concurrente y la programación lógica con restricciones. Se trata de un modelo de cómputo concurrente potente y sencillo basado en un *almacén* global, representado por un conjunto de restricciones sobre los valores que las variables pueden tomar, en vez de hallar una correspondencia entre valores y variables, lo cual provee solo información parcial sobre los valores que pueden tomar las variables. Todos los procesos de un sistema comparten este almacén común. En lugar de "leer" y "escribir" los valores de las variables, los procesos deben *chequear* (*ask*, comprobar si una restricción es deducible del almacén) y *propagar* (*tell*, añadir una nueva restricción al almacén).

Los lenguajes declarativos multiparadigma actuales [33, 53, 74] tratan de fusionar los paradigmas de programación declarativa más importantes, llamados programación funcional (estrategias de reducción eficientes, posibilidad de tener orden superior, etc.) y programación lógica con restricciones (resolución de objetivos, cómputos con información parcial, variables lógicas, etc.), con características concurrentes para la programación declarativa. Sin embargo, las interacciones entre todas esas características son muy complejas, de manera que el diseño concreto de una base teórica integrada de lenguajes y sistemas declarativos multiparadigma concurrentes no resulta nada trivial.

El esquema genérico de programación $CFLP(\mathcal{D})$ [52] captura de forma elegante las ideas fundamentales que hay detrás de estos sistemas y lenguajes, generalizando el esquema $CLP(\mathcal{D})$ sobre un dominio de restricciones dado paramétricamente para aportar fundamentos uniformes para las semánticas de lenguajes de programación funcionales y lenguajes de programación lógicos con restricciones. Las semánticas operacionales cuentan con un cálculo de estrechamiento dirigido por demanda con restricciones parametrizado por un resolutor de restricciones sobre un dominio $\mathcal{D}$ que es correcto y fuertemente completo con respecto a unas semánticas declarativas formalizadas por medio de una lógica de reescritura con restricciones, y mantiene las propiedades deseadas de la estrategia de estrechamiento necesario [5, 7, 21]. Además, $CFLP(\mathcal{D})$ es una base para la verificación declarativa [25] y la depuración

algorítmica de programas lógico funcionales con restricciones, y recientemente se ha expandido en varias direcciones, como por ejemplo, operaciones modulares de orden superior [65] y cooperación de dominios y resolutores de restricciones [26]. La versión actual del sistema lógico funcional con restricciones $\mathcal{TOY}$ (disponible en `toy.sourceforge.net`) y su lenguaje declarativo multiparadigma ha sido diseñado para implementar varias instancias del esquema $CFLP(\mathcal{D})$ y todas estas extensiones.

Sin embargo, la concurrencia no está aparentemente estudiada en el esquema de programación $CFLP(\mathcal{D})$, aunque existen formas declarativas de concurrencia para la programación lógico funcional pura. Por ejemplo [34] combina el estrechamiento necesario con el mecanismo de *residuación*, un enfoque alternativo para integrar la programación lógica y la programación funcional basado en la idea de retrasar o suspender las llamadas a funciones hasta que éstas estén listas para su evaluación, ofreciendo un principio de evaluación concurrente con sincronización por medio de las variables lógicas. Más recientemente, [28] presenta una extensión concurrente de programación lógico funcional donde la concurrencia es expresada por medio de procesos (o agentes) descrita como un *álgebra de procesos* [20, 31] como conceptos básicos, al mismo nivel que funciones o predicados.

A pesar todas estas extensiones de concurrencia para lenguajes de programación lógico funcionales, no tenemos conocimiento de ninguna investigación teórica correcta y completa sobre la combinación de estrechamiento dirigido por demanda con resolutores de restricciones cooperativos y residuación para aportar una integración declarativa más potente de técnicas de programación concurrente en el paradigma de la programación lógico funcional con restricciones. El desarrollo e implementación de todas esas técnicas es necesario para la implementación de sistemas declarativos multiparadigma, debido a que permiten optimizaciones relacionadas con la simplificación de los mecanismos de sincronización y comunicación que pueden reducir considerablemente el espacio de búsqueda. Por esta razón, creemos que un modelo declarativo para la concurrencia en el esquema genérico de programación lógico funcional con restricciones $CFLP(\mathcal{D})$ podría ser una extensión interesante para aportar una implementación eficiente del actual sistema lógico funcional con restricciones $\mathcal{TOY}$.

Además de este primer capítulo introductorio, este trabajo se divide en los siguientes capítulos:

- **Capítulo 2: Programación Lógica Concurrente con Restricciones.** Como ya hemos comentado anteriormente, los programas lógicos pueden interpretarse desde un punto de vista concurrente. Este capítulo es un estado del arte de este paradigma de programación, comenzando con la programación lógica pura, explicando sus principales características, como son el indetermin-

ismo y el algoritmo de unificación de las variables lógicas. A continuación introducimos el paradigma de la programación lógica con restricciones, las cuales aportan una mayor expresividad al lenguaje, junto con diferentes sistemas de restricciones concretos de utilidad práctica, como los sistemas de restricciones de *Herbrand* ($\mathcal{H}$) y de *Dominios Finitos* ($\mathcal{FD}$). Llegados a este punto, introducimos la programación lógica concurrente, como una interpretación concurrente para la programación lógica, y terminamos el capítulo explicando el esquema de la programación concurrente con restricciones.

- **Capítulo 3: Programación Lógico Funcional con Restricciones.** El modelo de evaluación perezosa ha demostrado ser muy eficiente dentro del paradigma de la programación funcional [16], además de permitir evaluar expresiones que no podían ser evaluadas por medio de la programación lógica, como el tratamiento de expresiones con listas potencialmente infinitas. En este capítulo estudiamos cómo se integra la programación funcional y la programación lógica con restricciones a través del esquema $CFLP(\mathcal{D})$ [52], permitiendo integrar las principales ventajas del paradigma de la programación funcional, como son la evaluación perezosa o el uso de elementos de orden superior. Además se introducirá el concepto de *árbol definicional con restricciones*, necesario para poder realizar de forma eficiente la evaluación perezosa de objetivos con variables lógicas y restricciones.

- **Capítulo 4: Programación Lógico Funcional Concurrente con Restricciones.** En este capítulo se plantea la parte más novedosa de nuestro trabajo, que consiste en la integración de las ideas expuestas en los dos capítulos anteriores, planteando un nuevo modelo de cómputo concurrente para el esquema $CFLP(\mathcal{D})$ basado en el uso de árboles definicionales para la sincronización mediante el uso de variables lógicas como canales de comunicación. Proporcionaremos una semántica operacional para el nuevo cálculo junto a sus propiedades de corrección y completitud.

- **Capítulo 5: Conclusiones y Trabajo Futuro.** En este último capítulo planteamos las principales conclusiones y las líneas de investigación que deja abiertos este trabajo. En particular, esbozaremos una posible implementación de nuestro sistema concurrente para el sistema de programación lógico funcional con restricciones $\mathcal{TOY}$, haciendo una comparativa de rendimientos de esta herramienta con y sin concurrencia, con respecto a otros lenguajes de programación lógico funcionales como *Curry* [33].

- **Apéndice A.** En este apéndice el lector puede encontrar las demostraciones de los principales resultados teóricos que se presentan en el trabajo, para una mejor legibilidad del texto principal del mismo.

Capítulo 2

Programación Lógica Concurrente con Restricciones

Comenzamos este trabajo introduciendo el paradigma de la *programación concurrente con restricciones*, que será una base fundamental para nuestro modelo operacional. En este capítulo se pretende ilustrar cómo se han añadido los principios de *paralelismo* y *concurrencia* a la *programación lógica* y a la *programación lógica con restricciones*, para que el lector interprete el cálculo propuesto en el Capítulo 4 de este trabajo como una combinación de ideas de estos paradigmas. Para ello, presentamos un estado del arte que sitúe al lector, comenzando desde la programación lógica, explicando los mecanismos de *unificación* y el uso de *variables lógicas* y un *modelo de cómputo indeterminista*, capaz de dar varias soluciones a un mismo problema. Continuaremos viendo cómo se han añadido las *restricciones* con el paradigma de la programación lógica con restricciones, viendo más en profundidad los sistemas de restricciones y varios ejemplos de ellos. Llegados a este punto, introduciremos el concepto de concurrencia, mostrando al lector el paradima de la programación lógica concurrente. Observaremos el paralelismo inherente que existe entre los cómputos en los programas lógicos y las *redes de procesos*, ilustrando una semántica operacional para dicho esquema, y dando la definición de *observable*, que representará el almacenamiento de resultados obtenidos por el cómputo. Finalizamos el capítulo viendo cómo se han añadido las restricciones al esquema anterior, dando el ejemplo del modelo de cómputo $CCP(\mathcal{C})$ y el sistema de transiciones $\mathcal{T}$, el cual será una base fundamental para nuestro cálculo en el Capítulo 4.

2.1 Programación Lógica

La *Programación Lógica* (LP, del inglés *Logic Programing*) [8, 15, 45, 47, 75] permite al investigador en Informática el uso de la lógica como lenguaje de programación, así como explorar modelos computacionales basados en deducciones controladas. Los

programas son entendidos como teorías de una lógica, de forma que la computación de un resultado use los mecanismos de deducción de dicha lógica con respecto a las teorías representadas por el programa.

La programación lógica se basa en la lógica de predicados, siendo común el uso de *Cláusulas de Horn*. Dichas cláusulas pueden emplearse como base para definir un lenguaje de programación gracias al uso de la *resolución SLD* (*Selective Linear Definite clause resolution*), la semántica operacional de las cláusulas de Horn, que puede implementarse de forma eficiente. Como semántica declarativa se usa una semántica basada en la teoría de modelos, tomando como dominio el *Universo de Herbrand*. La resolución *SLD* es un método de prueba por refutación, basado en un *algoritmo de unificación* [58], que permite vincular variables lógicas a valores. Las características de este tipo de programación pueden observarse mejor a través del siguiente ejemplo.

Ejemplo 1 (**Características de la Programación Lógica).** *Vamos a tratar el problema de la suma de dos números naturales. Definiremos la operación mediante el predicado "sum", el cual llevará tres operandos: los dos primeros referidos a los números que queremos sumar, y el tercer argumento, a la suma de los dos primeros. Usaremos una notación recursiva para los naturales, mediante la constante* 0*, para referirnos al número cero, y* $s(N)$ *para referirnos al sucesor de* N*. El programa resultante, expresado mediante cláusulas de Horn y mediante notación Prolog con las variables en mayúsculas, es el siguiente:*

$sum(0, X, X).$
$sum(s(X), Y, s(Z))$:- $sum(X, Y, Z).$

La primera cláusula afirma que la suma de dos naturales, siendo el primero cero y el segundo cualquier natural X*, es igual al segundo argumento* X*. La segunda cláusula afirma que, siendo* $s(X)$ *el primer argumento el sucesor de un número* X *y el segundo un número cualquiera* Y*, el resultado será el sucesor del resultado previo de sumar* X *e* Y*. El programa del ejemplo anterior ya es totalmente ejecutable como un programa lógico en Prolog [75]. Como se puede observar, no hay ninguna referencia a cómo se almacenan los datos en memoria, ni al manejo de la misma. En los lenguajes declarativos se automatizan los procesos de gestión de memoria para evitar fallos que son comunes en lenguajes imperativos debido, por ejemplo, al uso de punteros.*

Los lenguajes de programación lógica usan unificación para manejar variables libres en expresiones y para el cómputo de soluciones de variables libres. Esto permite que un mismo programa pueda responder a distintos objetivos sin necesidad de cambiar el programa. En el ejemplo anterior, podemos realizar las siguientes consultas (por comodidad expresaremos los naturales mediante su valor):

$$sum(4, 3, X). \qquad sum(4, Y, 7).$$

El primer objetivo nos devolverá en X el resultado de sumar 4 y 3, pero en el segundo objetivo la variable que queremos obtener es el segundo argumento, y el programa buscará el valor de Y tal que al sumárselo a 4 nos devuelva 7. Una característica fundamental de la programación lógica es el *indeterminismo* en la búsqueda de soluciones, que permite computar resultados parcialmente definidos para poder explorar todo el espacio de búsqueda. Como resultado inmediato, un mismo objetivo puede tener más de un resultado. Centrándonos en el ejemplo, vamos a ejecutar el siguiente objetivo:

$$sum(X, Y, 7).$$

A continuación vemos como trabajaría el mecanismo de resolución *SLD* para este objetivo. Cada flecha representa un paso de cómputo; el superíndice indica qué cláusula se escoge en cada caso y el subíndice indica el nivel en el árbol de resolución.

$sum(X, Y, 7) \Rightarrow_1^1 sum(0, 7, 7)[\boxed{X \mapsto 0, Y \mapsto 7}]$.
$\ldots \Rightarrow_1^2 sum(X_1, Y, 6)[X \mapsto s(X_1)] \Rightarrow_2^1 sum(0, 6, 6)[\boxed{X \mapsto 1, Y \mapsto 6}, X_1 \mapsto 0]$.
$\ldots \Rightarrow_2^2 sum(X_2, Y, 5)[X \mapsto s(X_1), X_1 \mapsto s(X_2)] \Rightarrow_3^1 sum(0, 5, 5)[\boxed{X \mapsto 2, Y \mapsto 5}, X_1 \mapsto 1, X_2 \mapsto 0]$
$\ldots$

Como primer resultado, obtenemos $X = 0$ e $Y = 7$, pero podemos ver que no es la única solución posible. Si seguimos pidiendo resultados, el programa buscará en el espacio de búsqueda definido por las cláusulas de Horn, dando los resultados $X = 1$ e $Y = 6$, $X = 2$ e $Y = 5$, etc.

2.2 Programación Lógica con Restricciones

La *Programación con Restricciones* (*CP*, del inglés *Constraint Programming*) [9] ha ido ganando importancia a lo largo de los últimos años al ser un paradigma basado en fundamentos teóricos muy sólidos [77], además de abarcar muchos campos de investigación, desde problemas teóricos hasta aplicaciones en la industria. La programación con restricciones se está haciendo cada vez más popular al ser una herramienta muy eficiente para resolver problemas de optimización que requieran resolución de restricciones heterogéneas y búsqueda combinatoria.

Las restricciones se usan para resolver problemas reales mediante la interacción de los diferentes componentes del problema. Así pues, una *restricción* será una relación entre dos componentes (por ejemplo, dos variables en *Prolog*). Un *Problema de*

Satistacción de Restricciones (*CSP*, del inglés *Constraint Satisfaction Problem*) es un conjunto de restricciones definidas sobre un número finito de variables. Resolver un *CSP* consiste en encontrar valores para las variables restringidas en un dominio de cómputo, de tal manera que se satisfagan todas las restricciones sobre las variables. Por ejemplo, la restricción X`#=`Y (notación de restricciones en *Prolog*) suponiendo X e Y restringidas al dominio de los booleanos $true$, $false$, tendría dos soluciones posibles: $X = true$ e $Y = true$, o $X = false$ e $Y = false$.

Las restricciones se han integrado en diferentes paradigmas de programación [52, 70], y se ha visto que los lenguajes de programación declarativos han resultado los más adecuados. En particular, las restricciones poseen una naturaleza relacional que hacen a los lenguajes lógicos los más apropiados para su integración.

La *Programación Lógica con Restricciones* (*CLP*, del inglés *Constraint Logic Programming*) ha recibido un gran impulso gracias a la aportación del esquema $CLP(\mathcal{C})$ de Jaffar y Lassez [40] como marco general para formalizar la semántica de programas lógicos parametrizado por un sistema de restricciones $\mathcal{C}$.

Los lenguajes de programación *CLP* nacen como una combinación entre el paradigma de la programación con restricciones y la programación lógica. La principal ventaja es que combina la declaratividad de los lenguajes lógicos con la eficiencia de *CP* [67]. Aunque los programas *CLP* dependen del dominio de aplicación, el esquema $CLP(\mathcal{C})$ permite la creación de lenguajes lógicos que comparten el mismo sistema de evaluación. En concreto, la idea básica del esquema $CLP(\mathcal{C})$ es la de reemplazar la unificación clásica de la programación lógica por un mecanismo de resolución de restricciones sobre un sistema de restricciones específico $\mathcal{C}$, denominado *resolutor de restricciones*.

Las diferentes instancias del esquema $CLP(\mathcal{D})$ se generan mediante las diferentes instancias de sistemas de restricciones $\mathcal{C}$ (reales, enteros, booleanos, conjuntos, etc.). Además, aportan expresividad al lenguaje, permitiendo resolver problemas de forma más sencilla que en *LP*. Para cada instancia de $\mathcal{C}$, el lenguaje *LP* se extiende mediante un conjunto de operaciones y mecanismos de resolución sobre dicho sistema disponibles para el programador. La idea fundamental del esquema *CLP* radica entonces en que tanto el lenguaje lógico subyacente como sus semánticas operacionales y declarativas pueden ser parametrizadas por un sistema de restricciones $\mathcal{C}$ y las operaciones específicas sobre ese sistema de restricciones.

2.2.1 Sistemas de Restricciones

Un *sistema de restricciones* $\mathcal{C}$ [69] viene dado por la siguiente tupla:

Hipótesis	$u, c \vdash c$
Corte	$\frac{u \vdash c \quad v, c \vdash d}{u, v \vdash d}$
$\wedge \vdash$	$\frac{u, c, d \vdash e}{u, c \wedge d \vdash e}$
$\vdash \wedge$	$\frac{u \vdash c \quad u \vdash d}{u \vdash c \wedge d}$
$\exists \vdash$	$\frac{u, c \vdash d}{u, \exists x.c \vdash d}$ Si X no es una variable libre ni de u ni d.
$\vdash \exists$	$\frac{u \vdash c[x/t]}{u \vdash \exists x.c}$

Figura 2.1: Reglas de inferencia para $\vdash$.

$$\mathcal{C} = (T, L, \vdash)$$

donde T es el conjunto de todos los términos que se construyen con operaciones del dominio del sistema de restricciones y variables de un conjunto infinito Var. L es el conjunto de las restricciones y fórmulas lógicas. Además, L se supone cerrado bajo los operadores de conjunción y existencial, es decir: $L = L^{\wedge,\exists}$. Los conjuntos T y L se suponen también cerrados bajo sustituciones $[x/t]$ con $x \in Var$ y $t \in T$.

$\vdash$ es la *relación de deducibilidad* y va desde las partes finitas de L hasta L (es decir, $\vdash \subseteq \mathcal{P}_{fin} \times L$). Dados $u \in \mathcal{P}_{fin}(L)$ y $c \in L$, la notación $u \vdash c$ (más precisamente $u \vdash_{\mathcal{C}} c$) indica que c se deduce de u en el sistema de restricciones $\mathcal{C}$. Además, la relación de deducibilidad es genérica con respecto a la sustitución:

$$u \vdash c \Rightarrow u[t/x] \vdash c[t/x] \ \forall x, t. x \in Var, t \in T$$

Se tiene que $\vdash$ satisface las reglas de inferencia que se describen en la tabla de la *Figura 2.1.*

Reflexividad	$u \vdash t = t$
Reemplazamiento	$\frac{u \vdash c\,[X/t]}{u, t{=}t' \vdash c\,[X/t']}$

Figura 2.2: Reglas de inferencia para $\vdash$ con igualdad.

- Las reglas de la tabla son válidas para $u \in \mathcal{P}_{fin}(L)$ y $c \in L$. Se escribe u, c para indicar la unión $u \cup \{c\}$, y no se descarta el caso en el que c esté incluido en u.
- Las reglas anteriores solo son válidas para u finito. Pero también es necesario contemplar el caso de conjuntos de restricciones potencialemente infinitos. Por ello, para $r, s \in \mathcal{P}_{fin}(L)$, se define también:
  - $r \vdash s$ si y solo si $r \vdash c$ para todo $c \in s$.
  - $r \vdash c$ si y solo si $u \vdash c$ para algún $u \in \mathcal{P}_{fin}(L)$.
- En el caso de tratarse de un sistema de restricciones con igualdad, debe cumplirse además que para cada par de términos $t, t' \in T$, la ecuación $t = t'$ pertenezca a L, y que además el operador $\vdash$ satisfaga otras dos reglas de inferencia descritas en la tabla de la *Figura 2.2.*

A partir de esta definición de relación de deducibilidad, se obtiene el siguiente resultado.

Proposición 2 (Propiedades de los sistemas de restricciones con igualdad). *Cualquier sistema de restricciones con igualdad, satisface las propiedades de simetría y transitividad.*

El lector puede encontrar la demostración de este resultado en el Apéndice A.

2.2.2 Consistencia y Satisfactibilidad

Un sistema de restricciones $\mathcal{C}$ permite reconocer la propiedad de *consistencia* si existe $F \in L$ que cumpla:

- $\nvdash F$ (Contradicción).
- $F \vdash c$ para cualquier $c \in L$.

Se dice entonces que F es una *restricción inconsistente*. Se puede postular también que $\top \in L$ (restricción trivial) que cumpla $\vdash \top$ y $\top \nvdash F$, pero no es necesario, ya que para cualquier $c \in L$, $\top \vdash c$ si y solo si $\vdash c$. Suponiendo que existe una restriccion inconsistente, definimos el predicado Con para el *chequeo de consistencia* de un conjunto de restricciones:

$$Con(r) \text{ si y solo si } r \nvdash F \text{ para } r \in \mathcal{P}(L).$$

Proposición 3 (Chequeo de consistencia). *Sean $c \in L$, $u, v \in \mathcal{P}_{fin}(L)$ y $r \in \mathcal{P}(L)$. Se verifica:*

- *$Con(r)$ si y solo si para todo $u \in \mathcal{P}_{fin}(L)$ se cumple $Con(u)$.*
- *Si $u \subseteq v$ y $Con(v)$ entonces $Con(u)$.*
- *Si $Con(u)$ y $u \vdash c$ entonces $Con(u \cup \{c\})$.*

Lema 4 (Lema de Consistencia). *Para cualquier sistema de restricciones $\mathcal{C} = (T, L, \vdash)$ y cualquier $u \in \mathcal{P}_{fin}(L)$, se verifica lo siguiente:*

$$\vdash \exists \overline{x}. \bigwedge u \Rightarrow Con(u)$$

para $\overline{x}$ definido como las variables locales de u. El recíproco es falso en general.

Corolario 5. *En las condiciones del lema anterior se tiene que:*

$$u \vdash F \Rightarrow \quad \nvdash \exists \bar{x}. \bigwedge u$$

Las demostraciones de estos resultados se dan en el Apéndice A.

2.2.3 Ejemplos de Sistemas de Restricciones

En esta subsección veremos algunos de los sistemas de restricciones más destacados sobre los que se centrará nuestro estudio.

El sistema de restricciones de Herbrand $\mathcal{H}$

En el sistema de restricciones de Herbrand los términos son de la forma:

$$t ::= X \mid f(t_1, ..., t_n)$$

Con $X \in Var$ y f un símbolo de función definida. Las restricciones atómicas L_0 serán igualdades o los valores de verdad *cierto* ($\top$) o *falso* (F):

$$c ::= t_1 = t_2 \mid T \mid F$$

El conjunto de las restricciones simples L_1 será el conjunto de las restricciones atómicas, sobre el que se ha aplicado el cierre con respecto a los operadores de conjunción y existencial, mientras que el conjunto de todas las restricciones L se obtiene al aplicarle el cierre sobre las operaciones de negación, conjunción, disyunción y cuantificadores universal y existencial:

$$L_1 = L_0^{\wedge,\exists}$$
$$L = L_0^{\neg,\wedge,\vee,\forall,\exists}$$

Definimos nuestro *operador de deducibilidad* para las restricciones de Herbrand:

$$u \vdash_{\mathcal{H}} c \Leftrightarrow_{def} CET \vdash \forall \bar{x}(\bigwedge u \rightarrow c) \text{ siendo } \bar{x} \text{ las variables locales de } c \text{ y } u.$$

y siendo CET los axiomas de la *Teoría de la Igualdad de Clark* [13, 57]:

- $\forall \bar{x} \forall \bar{y}.\, (f(\bar{x}) = f(\bar{y}) \rightarrow \bar{x} = \bar{y})$
- $\forall \bar{x} \forall \bar{y}.\, (f(\bar{x}) = g(\bar{y}) \rightarrow F)$ si $f \not\equiv g$
- $\forall x \forall \bar{y}.\, (x = t(x, \bar{y}) \rightarrow F)$ si $x \not\equiv t(x, \bar{y})$

Para el sistema de restricciones $\mathcal{H}$ se va a verificar el siguiente resultado, cuya demostración se puede encontrar en [56].

Teorema 6 (Mal'cew 1971 & Maher 1988). *CET es consistente, completa y decidible siempre que el conjunto de símbolos de función sea infinito.*

El sistema de restricciones para Dominios Finitos $\mathcal{FD}$

Para entender cómo son las restricciones de dominios finitos, primero hemos de definir los términos aritméticos y el concepto de rango. El conjunto de los términos aritméticos se define mediante la siguiente sintaxis:

$$t \ ::= \ X \mid n \ (n \in \mathbb{Z}) \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 * t_2 \mid t_1 \ div \ t_2 \mid t_1 \ mod \ t_2$$

Un *rango* no es más que un conjunto de valores del dominio en el que se está trabajando. Pueden definirse como valores intermedios entre dos términos aritméticos, y a su vez, aplicando los operadores de conjuntos sobre rangos:

$$r \ ::= \ [t_1..t_2] \mid [t..] \mid [t] \mid r_1 \cup r_2 \mid r_1 \cap r_2 \mid \backslash r_1 \mid r + t \mid r \ mod \ t$$

Ahora que ya hemos introducido el concepto de rango, podemos introducir el concepto de *restricción atómicas*. Una restricción atómica establece que una variable pertenece a un rango de valores. Así pues, L_0 es de la forma:

$$c ::= X \in r$$

El conjunto de restricciones corresponde al cierre del conjunto de las restricciones atómicas con respecto al operador de conjunción y al existencial:

$$L_1 = L_0^{\wedge,\exists}$$

En un sistema de restricciones de dominios finitos, la idea básica es ir añadiendo conjuntos a los que pertenece una variable, de cuya intersección obtenemos un rango cada vez más acotado (o vacío, en el caso de la intersección disjunta, lo que diría que nuestro sistema de restricciones sea *inconsistente*).

Dados $u \in \mathcal{P}_{fin}(L)$ y $c \in L$, definimos el *operador de deducibilidad* para restricciones con dominios finitos $\vdash_{\mathcal{FD}}$, donde denotaremos $\bar{x}$ a todas las variables locales de u y c:

$u \vdash_{\mathcal{FD}} c$ si y solo si $\forall \bar{x}.(\bigwedge u \to c)$ se cumple en la estructura con dominio $\mathbb{Z}$ y la interpretación estándar de los símbolos del lenguaje.

El sistema de restricciones $\mathcal{R}$

Para este sistema de restricciones, los términos son de la forma:

$$t ::= \ X \mid 0 \mid 1 \mid t_1 + t_2 \mid t_1 * t_2$$

Las restricciones atómicas L_0 serán relaciones definidas entre términos, de igualdad, orden, y orden estricto, tal como se expresa a continuación:

$$c ::= \ t_1 = t_2 \mid t_1 \leq t_2 \mid t_1 < t_2$$

Los conjuntos de las restricciones simples, y todas las restricciones, se obtienen aplicando el cierre de operadores de la misma manera que para los sistemas de restricciones de Herbrand:

$$L_1 = L_0^{\wedge,\exists}$$
$$L = L_0^{\neg,\wedge,\vee,\forall,\exists}$$

Dados $u \in \mathcal{P}_{fin}(L)$ y $c \in L$, definimos el *operador de deducibilidad* $\vdash_{\mathcal{R}}$ de la siguiente manera:

$u \vdash_{\mathcal{R}} c$ si y solo si $\forall \bar{x}.(\bigwedge u \to c)$ se cumple en el cuerpo de los números reales.

En particular se tiene que la anterior definición es equivalente a que "$\forall \bar{x}.(\bigwedge u \to c)$" se cumple en cualquier cuerpo real cerrado.

Se cumple también que $\vdash_{\mathcal{R}}$ es decidible (Véase [76]).

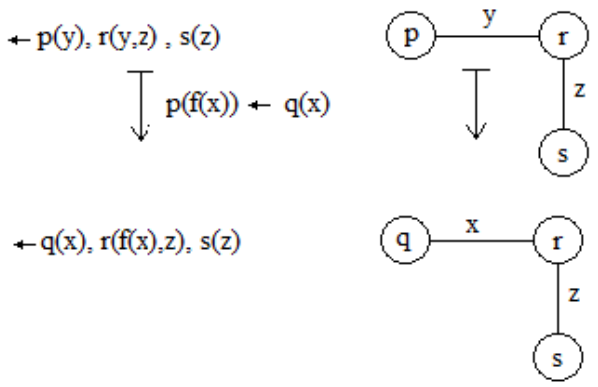

Figura 2.3: Paralelismo entre programas lógicos y redes de procesos.

2.3 Programación Lógica Concurrente

En esta sección vamos a introducir los primeros conceptos de concurrencia de este trabajo, aplicados al paradigma de la programación lógica. Se ilustrará primero el paralelismo existente entre un programa lógico y una red de procesos. A continuación pasaremos a introducir una semántica operacional que represente los programas lógicos como redes de procesos y finalmente analizaremos sus resultados de corrección y completitud.

2.3.1 Redes de Procesos Paralelos en Programación Lógica

Analizando la ejecución de los programas lógicos, podemos ver que su estructura parece adecuada para realizar una interpretación de los programas lógicos como *redes de procesos paralelos* [73, 78]. Un objetivo atómico $p(E)$ sería considerado como un *proceso* de nombre p y E los datos que lleva, y un objetivo compuesto como una red de procesos paralelos. Las variables compartidas se entienden como *canales de comunicación* entre variables, comunicándose los procesos al vincular un valor a una variable. Un paso de resolución en un programa lógico puede cambiar uno o más objetivos atómicos, lo que reflejaría una *reconfiguración* de la red de procesos. Bajo esta *interpretación concurrente*, los programas lógicos pueden verse como definiciones de procesos.

Podemos apreciar más fácilmente las similitudes a través del ejemplo mostrado en la *Figura 2.3*, que representa un objetivo lanzado.

El objetivo $\leftarrow p(y), r(y,z), s(y)$ puede representarse mediante una red de procesos como la que se ve en la *Figura 2.3*, donde cada subobjetivo $p(y)$, $r(y,z)$ y $s(y)$ puede ser interpretado como un proceso independiente, y las variables y y z como canales de comunicación entre dichos procesos. Vemos como un paso de resolución del objetivo atómico $p(y)$ queda reflejado como un cambio en la red de procesos, cambiando el nombre del proceso a q. Además, al unificar y con $f(x)$, el proceso p (renombrado como q) le envía información a r.

Ejemplo 7. *Podemos hacer un acercamiento más detallado estudiando el comportamiento paralelo de un programa lógico como el de los números de Fibonacci, una función recursiva que recibe un argumento n de tipo natural, y devuelve* 1 *si* $n = 0$ *o* $n = 1$*, y para* $n \geq 2$ *devuelve la suma de los números de fibonacci de* $n-1$ *y* $n-2$*. Usaremos el símbolo "*$\leftarrow$*" en lugar de "*:-*":*

$fib(0, s(0)) \leftarrow .$
$fib(s(0), s(0)) \leftarrow .$
$fib(s(s(X)), Y) \leftarrow fib(s(X), Y1), fib(X, Y2), sum(Y1, Y2, Y).$

Lanzamos el objetivo $\leftarrow fib(s^3(0), y)$*, cuya resolución reflejamos mediante la reconfiguración de la red de procesos descrita en la* Figura 2.4*.*

Hay un posible paralelismo entre cada uno de los subobjetivos, puesto que Y1 no necesita hacer nada referente al cálculo de $Y2$ *y viceversa. En el caso de que hubiera variables compartidas, éstas serían vistas como canales de comunicación. Además, se aprecia la necesidad de combinar respuestas. En este ejemplo, para calcular* $\leftarrow fib(s^3(0), Y)$*, primero deben resolverse los subobjetivos* $\leftarrow fib(s^2(0), Y1)$ *y* $\leftarrow fib(s(0), Y2)$*, y las respuestas se combinan con* $sum(Y1, Y2, Y)$*, obteniendo así el resultado.*

Uso de Variables Compartidas

Como se ha visto anteriormente, las variables compartidas entre objetivos atómicos se modelan como canales de comunicación entre procesos. Tomando una variable arbitraria que se comporte como un canal de comunicación, llamemosla X, decimos que X recibe un mensaje al vincularse a un valor o término t en la forma $X = t$.

Es necesario pensar en la posibilidad de que un canal reciba una sucesión potencialmente infinita de mensajes, que permita enviar tantos mensajes como requiera la realización de la tarea de los procesos. Esto se consigue si el término t contiene otra variable, que será usada como nuevo canal con información añadida. Una posible manera de representar esta situación es usando una estructura de lista:

$$X = [t|Xs]$$

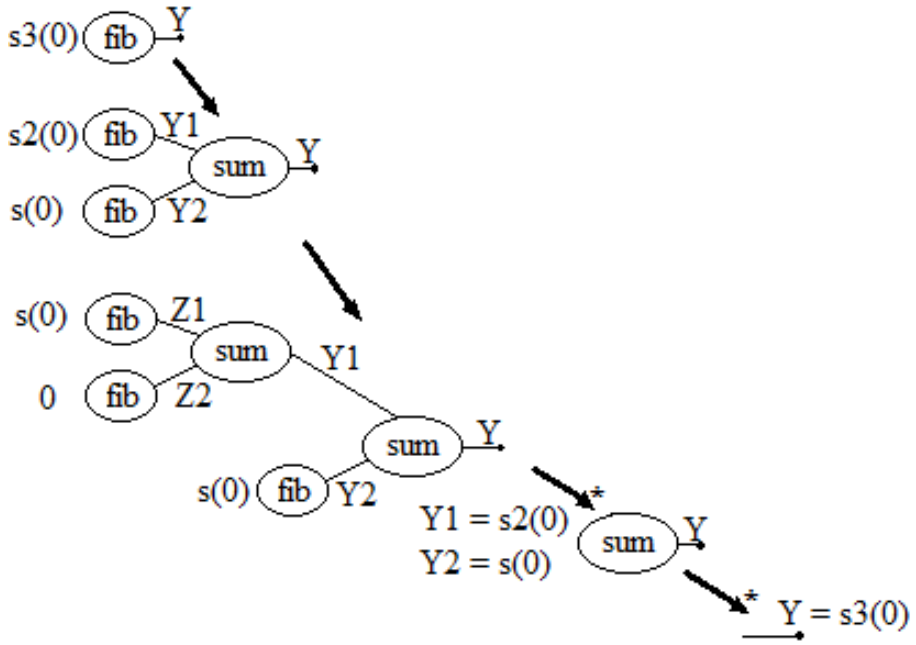

Figura 2.4: Pasos de resolución de la llamada Fibonacci.

Se consigue así el uso de la combinación de listas potencialmente infinitas (o *streams*) como canales de comunicación. Un canal que ya ha recibido n mensajes sería de la forma:

$$[t_1, t_2, \ldots, t_n | Xs]$$

Semántica Operacional

Llegados a este punto, es necesario definir una semántica operacional para la interpretación concurrente de un programa lógico. Definiremos una *configuración* como un par, donde la primera componente G es un *objetivo* (una red de procesos desde el punto de vista concurrente), y θ una *sustitución* (vínculos acumulados):

$$\langle\, G \,|\, \theta \,\rangle$$

Antes de pasar a describir la semántica operacional, recogemos algunas definiciones previas relativas a sustituciones que van a ser necesarias.

Definición 8 (**Sustituciones).** *Sea $T(\Sigma, \mathcal{V})$ un conjunto de términos construidos a partir de una signatura Σ y un conjunto de variables $\mathcal{V}$. Una sustitución es una función $\sigma : \mathcal{V} \to T(\Sigma, \mathcal{V})$ que verifica que $\sigma(v) \neq v$ para un número finito de variables en $\mathcal{V}$; el conjunto de esas variables se denomina dominio y se denota por $Dom(\sigma)$. Si $Dom(\sigma) = \{x_1, \ldots, x_n\}$ y $\forall i \in \{1, \ldots, n\}.\sigma(x_i) = t_i$, entonces σ puede representarse como $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$.*

Definición 9 (**Aplicación de una sustitución a un término).** *Sea $t \in T(\Sigma, \mathcal{V})$ un término, y $\sigma : \mathcal{V} \to T(\Sigma, \mathcal{V})$ una sustitución. La aplicación de σ a t se escribe $\sigma(t)$ y se define por inducción en la estructura de t:*

- *Si $t \in \mathcal{V}$, $\sigma(t) = \sigma(t)$.*
- *Si $t \in \Sigma^0$, $\sigma(t) = t$.*
- *Si $t = f(t_1, \ldots, t_n)$, y la aridad de f es n, con $n > 0$ y $\sigma(f(t_1, \ldots, t_n))$=$f(\sigma(t_1), \ldots, \sigma(t_n))$.*

Ejemplo 10 (**Relación de orden entre sustituciones).** *Vamos a definir tres sustituciones para un sistema con $\mathcal{V} = \{X, Y, Z\}$:*

- $\sigma_1 = \{X \mapsto Z\}$
- $\sigma_2 = \{X \mapsto Z\, Y \mapsto s(0)\}$
- $\sigma_3 = \{Y \mapsto 0, Z \mapsto s(X)\}$

Tenemos que σ_1 es más general *que σ_2 (representado por $\sigma_1 \leq \sigma_2$), porque $\sigma_2 = \{Y \mapsto s(0)\}\sigma_1$. No podemos comparar con esta relación de orden a σ_2 y σ_3 porque $\sigma_2(Y) = s(0)$ y $\sigma_3(Y) = 0$. No podemos encontrar una sustitución ρ tal que $\rho(0) = s(0)$ o $\rho(s(0)) = 0$. La ordenación de sustituciones es un orden parcial, por lo que no todo par de sustituciones pueden compararse entre sí.*

Definición 11 (**Unificación).** *Dos términos s y t son* unificables *si existe una sustitución σ tal que $\sigma(s) \equiv \sigma(t)$. Entonces σ se llama* unificador *de s y t. Si σ verifica que para todo unificador θ de s y t, $\sigma \leq \theta$, entonces σ es el* unificador más general (u.m.g.) *de s y t.*

Las *reglas de transición* que definen la semántica operacional para el lenguaje se muestran en la tabla de la *Figura 2.5* y se explican a continuación:

La interpretación de la regla **C1** es que un proceso independiente realizará un paso de resolución según su definición en el programa $\mathcal{P}$. La regla **C2** permite que una red de procesos progrese si un subconjunto de procesos puede progresar. Estas dos reglas nos van a permitir tratar la concurrencia para programas lógicos.

C1	$\langle \leftarrow A \mid \theta \rangle \Rightarrow \langle \leftarrow \bar{B} \mid \theta\sigma \rangle$ Si existe en el programa $\mathcal{P}$ una regla $A' \leftarrow \bar{B}$ tal que $\sigma = u.m.g.(A\theta, A')$.
C2	$\dfrac{\langle \leftarrow \bar{A} \mid \theta \rangle \Rightarrow \langle \leftarrow \bar{A'} \mid \theta' \rangle}{\langle \leftarrow \bar{B}, \bar{A}, \bar{C} \mid \theta \rangle \Rightarrow \langle \leftarrow \bar{B}, \bar{A'}, \bar{C} \mid \theta' \rangle}$

Figura 2.5: Reglas de transición para programas lógicos interpretados como redes de procesos paralelos.

Usaremos la notación $\langle A \mid \theta \rangle \Rightarrow \langle A' \mid \theta' \rangle$ para indicar que se puede pasar de la configuración $\langle A \mid \theta \rangle$ a $\langle A' \mid \theta' \rangle$ en un paso aplicando las reglas de transición. Usaremos $\Rightarrow^*$ para representar el cierre reflexivo transitivo de $\Rightarrow$. Además $\langle A \mid \theta \rangle \not\Rightarrow$ indica que no se puede aplicar ninguna regla de transición para resolver el objetivo.

En el siguiente ejemplo se verá cómo se aplican las reglas de la semántica operacional para resolver un objetivo concreto.

Ejemplo 12. *Vamos a resolver el objetivo* $\leftarrow fib(s^3(0), Y)$ *para el programa del* Ejemplo 7*, suponiendo definido el predicado* $sum(X, Y, Z)$*, que devuelve en el último argumento la suma de los dos primeros.*

$\langle \leftarrow \underline{fib(s(s(s(0))), Y)} \mid \{\} \rangle$
$\Rightarrow_{C1} \langle \leftarrow fib(s(s(0)), Y1), \underline{fib(s(0), Y2)}, sum(Y1, Y2, Y) \mid \{\} \rangle$
$\Rightarrow_{C1,C2} \langle \leftarrow \underline{fib(s(s(0)), Y1)}, sum(Y1, Y2, Y) \mid \{Y2 \mapsto s(0)\} \rangle$
$\Rightarrow_{C1,C2} \langle \leftarrow \underline{fib(s(0), Y1')}, fib(0, Y2'), sum(Y1', Y2', Y1), sum(Y1, Y2, Y) \mid \{Y2 \mapsto s(0)\} \rangle$
$\Rightarrow_{C1,C2} \langle \leftarrow \underline{fib(0, Y2')}, sum(Y1', Y2', Y1), sum(Y1, Y2, Y) \mid \{Y2 \mapsto s(0), Y1' \mapsto s(0)\} \rangle$
$\Rightarrow_{C1,C2} \langle \leftarrow \underline{sum(Y1', Y2', Y1)}, sum(Y1, Y2, Y) \mid \{Y2 \mapsto s(0), Y1' \mapsto s(0), Y2' \mapsto s(0)\} \rangle$
$\Rightarrow^* \langle \leftarrow \underline{sum(Y1, Y2, Y)} \mid \{Y1 \mapsto s(s(0)), Y2 \mapsto s(0), Y1' \mapsto s(0), Y2' \mapsto s(0)\} \rangle$
$\Rightarrow^* \langle \leftarrow \mid \{\boxed{Y \mapsto s(s(s(0)))}, Y1 \mapsto s(s(0)), Y2 \mapsto s(0), Y1' \mapsto s(0), Y2' \mapsto s(0)\} \rangle$

Partiendo del objetivo inicial $\langle \leftarrow \underline{fib(s(s(s(0))), Y)} \mid \{\} \rangle$ *aplicando las reglas de cómputo llegamos al objetivo final en forma resuelta* $\langle \leftarrow \mid \{Y \mapsto s(s(s(0))), Y1 \mapsto s(s(0)), Y2 \mapsto s(0), Y1' \mapsto s(0), Y2' \mapsto s(0)\} \rangle$*, donde queda almacenado el resultado en la variable* Y*, es decir, el valor* $s(s(s(0)))$*, que es el resultado de aplicar la función de fibonacci al argumento* $s(s(s(0)))$*.*

Observables y Composicionalidad

Al ejecutar un objetivo sobre un programa lógico funcional, el cómputo puede escoger varias ramas de ejecución de forma indeterminista. Aplicando concurrencia se espera añadir eficiencia en el cálculo de una única solución, de forma que se puedan computar concurrentemente y combinar las diferentes soluciones de la evaluación indeterminista. Para ello, debemos dar algunas definiciones previas.

Definición 13 (**Observables de éxito**). *Para un programa $\mathcal{P}$ dado, definimos el conjunto de observables de éxito con respecto a un objetivo G como el conjunto $\mathcal{O}_{SS}(G)$ de todas las posibles configuraciones que terminan con éxito devolviendo las sustituciones de las variables para las soluciones encontradas. Formalmente:*

- $\mathcal{O}_{SS}(G) =_{def} \{\theta \restriction_G \mid \langle \leftarrow G \mid \{\}\rangle \Rightarrow^* \langle \{\} \mid \theta\rangle\}$
- $SS =_{def} \{A \mid A \ cerrado, \{\} \in \mathcal{O}_{SS}(\leftarrow A)\}$
- $\epsilon(\theta) =_{def} \{x = t \mid x \in dom(\theta), \theta(x) = t\}$
- $\theta \,||\, \sigma =_{def} u.m.g.(\epsilon(\theta) \cup \epsilon(\sigma))$
- $ff =_{def}$ *resultado de una rama de cómputo que falla.*

donde:

- *Un objetivo A se considera* cerrado *cuando no hay apariciones de variables libres.*
- *El operador $\restriction_G$ aplicado a la sustitución θ, restringe el dominio de θ a las variables que aparecen en el objetivo G.*

El siguiente resultado, cuya demostración puede encontrarse en [19], permite demostrar la composicionalidad del conjunto de observables.

Proposición 14 (**de Boer & al. 1992**). *La definición que se ha dado de $\mathcal{O}_{SS}$ es composicional. Se tiene que para todo objetivo compuesto $\leftarrow \bar{A}, \bar{B}$:*

$$\mathcal{O}_{SS}(\leftarrow \bar{A}, \bar{B}) = \mathcal{O}_{SS}(\leftarrow \bar{A}) \;\widetilde{||}_{SS}\; \mathcal{O}_{SS}(\leftarrow \bar{B})$$

donde para X e Y conjuntos de sustituciones:

$$X \,\widetilde{||}_{SS}\, Y =_{def} \{\, \theta \,||\, \sigma \mid \theta \in X, \sigma \in Y, \theta \,||\, \sigma \neq ff\}$$

Ahora que se ha definido una *semántica operacional composicional*, queda justificada la afirmación de que los programas lógicos son inherentemente paralelos. Esto se cumple porque el estado θ evoluciona monótonamente, y porque no hay mecanismos de suspensión de procesos.

Definición 15 (Observables de fallo). *Una vez definidos los observables de éxito y visto que su definición es composicional, se definen el conjunto de observables de fallo como aquellas ramas de cómputo que no han podido completarse al no ser válida la solución. Para un programa $\mathcal{P}$ dado se define:*

- $\mathcal{O}_{ff}(G) =_{def} \{ff \mid$ *todo cómputo a partir de* G *de la forma* $\langle G \mid \{\}\rangle \Rightarrow^*$ $\langle G' \mid \theta\rangle \not\Rightarrow$ *siendo* G' *no vacío*$\}$.

- $FF =_{def} \{A \mid A$ *cerrado*, $ff \in \mathcal{O}_{ff}(\leftarrow A)\}$.

Con esta definición de observable de fallo obtenemos dos observaciones inmediatas: la primero, el observable de fallo de un objetivo siempre será que, o existe fallo (ff), o no existe, y la segundo, que el hecho de que no existan observables de éxito no implica que se de un fallo, puesto que pueden haber ramas de cómputo infinito no contempladas por ninguno de los dos observables:

- $\mathcal{O}_{ff}(G) = \{ff\}$ ó $\mathcal{O}_{ff}(G) = \{\}$.

- $\mathcal{O}_{SS}(G) = \{\} \not\Rightarrow ff \in \mathcal{O}_{ff}(G)$.

La definición dada para los observables de fallo no es composicional con respecto a la de los observables de éxito, es decir, no existe ningún operador de composición $\widetilde{\|}_{ff}$ que cumpla:

$$\mathcal{O}_{ff}(\leftarrow \bar{A}, \bar{B}) = \langle \mathcal{O}_{SS}(\leftarrow \bar{A}), \mathcal{O}_{ff}(\leftarrow \bar{A})\rangle \;\widetilde{\|}_{ff}\; \langle \mathcal{O}_{SS}(\leftarrow \bar{B}), \mathcal{O}_{ff}(\leftarrow \bar{B})\rangle$$

Ejemplo 16. *Con el siguiente programa veremos como la definición de* $\widetilde{\|}_{ff}$ *no es composicional. Sea un programa lógico* $\mathcal{P}$ *definido con las siguientes cláusulas de Horn:*

$p(a) \leftarrow p(a).$
$q(b) \leftarrow q(b).$
$r(a) \leftarrow .$

Considerando los objetivos atómicos $A = p(x)$, $B = q(x)$ *y* $C = r(x)$, *se tiene que:*

- $\mathcal{O}_{SS}(\leftarrow A) = \mathcal{O}_{SS}(\leftarrow B) = \{\}$

- $\mathcal{O}_{ff}(\leftarrow A) = \mathcal{O}_{ff}(\leftarrow B) = \{\}$

- $\mathcal{O}_{ff}(\leftarrow A, C) = \{\} \neq \{ff\} = \mathcal{O}_{SS}(\leftarrow B, C)$

Se puede ver que tanto los observables de éxito y fracaso de los objetivos atómicos A y B son el conjunto vacío, puesto que al tratar de resolver el objetivo entra en una rama de cómputo infinita.

Como puede verse en el ejemplo anterior, es necesario tener en cuenta las ramas de cómputo infinitas a la hora de dar una definición composicional para los observables de fallo. [30, 46] da la siguiente definición para observables de cómputos infinitos:

$$\mathcal{O}_{ii}(G) =_{def} \{\ lim_n \theta_n \restriction_G \ \mid \langle G \mid \{\}\rangle \Rightarrow \langle G_1 \mid \theta_1\rangle \Rightarrow \ldots \langle G_n \mid \theta_n\rangle \ldots \text{ infinita y equitativa } \}$$

Proposición 17 (Composicionalidad de $\mathcal{O}_{ff}, \mathcal{O}_{ii}$). *Sea $\mathcal{O}(G) =_{def} \mathcal{O}_{SS}(G) \cup \mathcal{O}_{ff}(G) \cup \mathcal{O}_{ii}(G)$. Obtenemos composicionalidad para los observables de fallo:*

- $\mathcal{O}_{ff}(\leftarrow \bar{A}, \bar{B}) = \mathcal{O}(\leftarrow \bar{A})\ \tilde{\|}_{ff}\ \mathcal{O}(\leftarrow \bar{B})$ *siendo* $\tilde{\|}_{ff}$ *de la siguiente manera:*
  $X\ \tilde{\|}_{ff}\ Y =_{def} \{ff \mid ff \in X \cup Y \quad o \quad \forall \theta \in X. \forall \sigma \in Y.\ \theta \| \sigma = ff\}$
- $\mathcal{O}_{ii}(\leftarrow \bar{A}, \bar{B}) = \langle \mathcal{O}_{SS}(\leftarrow \bar{A}), \mathcal{O}_{ii}(\leftarrow \bar{A})\rangle \tilde{\|}_{ii} \langle \mathcal{O}_{SS}(\leftarrow \bar{B}), \mathcal{O}_{ii}(\leftarrow \bar{B})\rangle$

Definiendo el operador de composición $\|_{ii}$ de la siguiente manera:

$$\langle X_1, X_2\rangle \tilde{\|}_{ii} \langle Y_1, Y_2\rangle =_{def} \{\theta \| \sigma \mid \theta \in X_2, \sigma \in Y_1 \cup Y_2, \theta \| \sigma \neq ff\} \cup \{\theta \| \sigma \mid \theta \in X_1 \cup X_2, \sigma \in Y_2, \theta \| \sigma \neq ff\}$$

Ejemplo 18. *Ahora que tenemos un nuevo operador de composición para observables de fallos que tiene en cuenta las ramas de cómputo infinitas, en el programa del* Ejemplo 16 *se sigue cumpliendo lo siguiente:*

$$\mathcal{O}_{ff}(\leftarrow A, C) \neq \mathcal{O}_{ff}(\leftarrow B, C)$$

Pero ésto se explica porque tenemos distintos observables de cómputo infinito:

$$\mathcal{O}_{ii}(\leftarrow A) = \{\{x/a\}\} \neq \{\{x/b\}\} = \mathcal{O}_{ii}(\leftarrow B)$$

2.3.2 Concurrencia en *CLP*

En esta subsección vamos a presentar el esquema $ALPS(\mathcal{C})$ [43, 55] (del inglés *Annotated Logic Programming Systems*), que es el resultado de añadir mecanismos de sincronización declarativa al esquema $CLP(\mathcal{C})$. Cada instancia del esquema *ALPS* se

basa en un sistema de restricciones $\mathcal{C}$, con una teoría asociada, existencialmente completa. Además, cada sistema de restricciones aportará sus operaciones específicas.

Para cada lenguaje de restricciones se supone que la ecuación de igualdad $t = t'$ es una primitiva del lenguaje, y además, el lenguaje es un conjunto cerrado bajo las operaciones de conjunción, existencial, y sustitución. Las cláusulas en un programa *ALPS* son de la forma:

$$A \leftarrow \bar{c} \,|\, \bar{B}$$

donde A es la cabeza de la cláusula (o átomo), $\bar{c}$ es la guarda que se ha de comprobar, siento ésta una conjunción de restricciones, y $\bar{B}$ es el cuerpo de la cláusula que se ejecuta una vez se ha comprobado la guarda, y es una conjunción de átomos y restricciones. Con este tipo de cláusulas, las configuraciones o estados de cómputo son ahora de la forma:

$$<\leftarrow \bar{A} \,|\, \bar{c}>$$

donde $\bar{A}$ representa la red de procesos como conjunción de átomos y restricciones, y $\bar{c}$ es la conjunción de restricciones almacenadas.

Sincronización Declarativa en *ALPS*

Se basa en la relación lógica entre el almacén de restricciones $\bar{c}$, la cabeza atómica A, que se va a reducir, y la cláusula candidata a efectuar dicha reducción $A' \leftarrow \bar{c}' \,|\, \bar{B}$, que deberá formar parte del programa $\mathcal{P}$.

Sean $\bar{x}$ las variables locales de A' y $\bar{c}'$, definimos:

(a) $\bar{c}$, A *satisfacen* la cláusula si $\vdash_{\mathcal{C}} \exists\bar{x}.(\bar{c} \wedge A = A' \wedge \bar{c}')$.

(b) $\bar{c}$, A *validan* la cláusula si $\bar{c} \vdash_{\mathcal{C}} \exists\bar{x}.(A = A' \wedge \bar{c}')$.

(c) $\bar{c}$, A *invalidan* la cláusula si $\vdash_{\mathcal{C}} \neg\exists\bar{x}.(\bar{c} \wedge A = A' \wedge \bar{c}')$.

En particular, se tiene que cuando $\mathcal{C}$ es la *Teoría de Clark* (Vease *Subsección 2.2.3*) para el sistema de restricciones de Herbrand se obtienen los siguientes resultados:

(a) es equivalente a afirmar que A *unifica* con A'.

(b) es equivalente a afirmar que A *es instancia* de A'.

(c) es equivalente a afirmar que la unificación de A con A' *falla*.

C1	$\langle \leftarrow c \mid \bar{c} \rangle \Rightarrow \langle \{\} \mid \bar{c}, c \rangle$ si c es una restricción y $\vdash_{\mathcal{C}} \exists \bar{x}.(\bar{c} \wedge c)$.
C2a	$\langle \leftarrow A \mid \bar{c} \rangle \Rightarrow \langle \leftarrow \bar{B} \mid \bar{c}, A = A', \bar{c}' \rangle$ si A es un átomo y existe en el programa una cláusula de la forma $A' \leftarrow \bar{c}' \mid \bar{B}$ tal que ésta es la única cláusula *satisfecha* por $\bar{c}$, A.
C2b	$\langle \leftarrow A \mid \bar{c} \rangle \Rightarrow \langle \leftarrow \bar{B} \mid \bar{c}, A = A', \bar{c}' \rangle$ si A es un átomo y existe en el programa una cláusula de la forma $A' \leftarrow \bar{c}' \mid \bar{B}$ tal que es *validada* por $\bar{c}$, A.
C3	$\dfrac{\langle \leftarrow \bar{A} \mid \bar{c} \rangle \Rightarrow \langle \leftarrow \bar{A}' \mid \bar{c}' \rangle}{\langle \leftarrow \bar{B}, \bar{A}, \bar{C} \mid \bar{c} \rangle \Rightarrow \langle \leftarrow \bar{B}, \bar{A}', \bar{C} \mid \bar{c}' \rangle}$

Figura 2.6: Reglas de cómputo para $ALPS(\mathcal{C})$.

Reglas de Cómputo para *ALPS*

Una vez definida la estructura del lenguaje *ALPS*, los mecanismos de sincronización declarativa, y los estados de cómputo, pasamos a describir las *reglas de transición de estados* expuestas en la *Figura 2.6* y en [69].

La regla **C1** trata el caso en el que el paso de cómputo requiera el tratamiento de una restricción, la cual pasará al almacén de restricciones. Hay que tener en cuenta que, aunque las reglas **C2a** y **C2b** se solapen, no generan contradicciones, puesto que si $\bar{c}, A$ validan una cláusula, también la satisfacen (suponiendo $\bar{c}$ satisfactible), y ninguna otra cláusula puede ser la única satisfecha. Por otra parte, si $\bar{c}$ y A no validan ninguna cláusula, entonces **C2b** no es aplicable. En consecuencia, solo quedan dos casos, o bien **C2a** es aplicable y elige una única cláusula, o bien A queda suspendido.

La regla **C3** expresa la composición paralela mediante entrelazado, y es la que permite que un sistema avance aunque algunos de sus elementos estén bloqueados.

Estas reglas aportan mucha expresividad al lenguaje *ALPS*, siendo más expresivo que *P-Prolog* [80] y que *GHC* (del inglés *Guarded Horn Clauses*) [79]. Lo vemos a través del siguiente ejemplo:

Ejemplo 19. *Tomemos el programa que concatena dos listas:*

$$append([\,], Ys, Ys) \leftarrow |.$$
$$append([X|Xs], Ys, [X|Zs]) \leftarrow |append(Xs, Ys, Zs).$$

*Algunos objetivos se bloquearían en GHC, pero tienen éxito en ALPS gracias a la regla **C2a**:*

$$\langle \leftarrow append(Xs, Ys, [\,]) \mid \rangle \Rightarrow_{C2a} \langle \{\} \mid Xs = [\,], Ys = [\,] \rangle$$

Corrección y Completitud en $ALPS(\mathcal{C})$

Obtener corrección para el cómputo de respuestas es de vital importancia, puesto que será lo que nos asegure que todas las respuestas que nos de un programa $ALPS(\mathcal{C})$ sean correctas, es decir, no de ningún resultado incorrecto. El siguiente teorema demuestra la corrección del lenguaje, y se basa en la *Teoría de Complección de Clark*.

Dado un programa $\mathcal{P}$ escrito en $CLP(\mathcal{C})$, calcular su *complección de Clark* $\mathcal{P}^*$ consiste en sustituir todos los términos que aparecen en la cabeza de las cláusulas por variables nuevas y distintas entre sí, añadiendo a la comprobación de la guarda las restricciones de igualdad de las nuevas variables con los términos que se sustituyen, respectivamente:

$$p(t_1, \ldots, t_n) \leftarrow \bar{G} \,|\, \bar{B}$$
$$\downarrow$$
$$p(X_1, \ldots, X_n) \leftarrow X_1 = t_1, \ldots, X_n = t_n, \bar{G} \,|\, \bar{B}$$

Se cumple entonces el siguiente resultado de corrección:

Teorema 20 (Corrección). *Si dado un objetivo existe un número de pasos de cómputo tal que se llega a una configuración donde no quedan subobjetivos por resolver, entonces decimos que el conjunto de restricciones obtenidas satisface el objetivo, para la complección de Clark $\mathcal{P}^*$ de un programa $\mathcal{P}$, y bajo un sistema de restricciones $\mathcal{C}$. Es decir, si $\langle G \,|\, \{\}\rangle \Rightarrow^* \langle \{\} \,|\, \bar{c}\rangle$ entonces $\bar{c} \vdash_{\mathcal{C},\mathcal{P}^*} G$.*

La completitud es otro aspecto importante a tener en cuenta; es la propiedad que nos garantiza que dado un objetivo, el programa nos devolverá todas sus posibles soluciones. Para $CLP(\mathcal{C})$ se tiene el siguiente teorema de completitud:

Teorema 21 (Completitud fuerte de las respuestas calculadas con éxito). *Si se tiene que* $\vdash_{\mathcal{C}} \exists \bar{x}.\bar{c}$*, siendo* $\bar{x}$ *las variables locales en* c*, y* $\bar{c} \vdash_{\mathcal{C},\mathcal{P}^*} G$*, entonces se verifica que existen* k *cómputos con éxito de la forma* $\langle G \mid \{\}\rangle \Rightarrow^* \langle \{\} \mid \vec{c}_i'\rangle (1 \leq i \leq k)$ *tales que* $\bar{c} \vdash_{\mathcal{C}} \bigvee_{i=1}^{k} \exists \bar{y}_i.\vec{c}_i'$*, donde* $\vec{c}_i'$ *se entiende como conjunción de restricciones, e* $\bar{y}_i$ *son las variables locales en* $\vec{c}_i'$*.*

Este teorema es falso en general para $ALPS(\mathcal{C})$, debido a la *elección con compromiso*. Lo ilustramos a través del siguiente ejemplo:

Ejemplo 22. *Tomemos como ejemplo el siguiente programa* $\mathcal{P}$ *que define un predicado "merge" escrito en* $ALPS(\mathcal{H})$*, que realiza la mezcla de dos listas de forma indeterminista.*

$$
\begin{array}{l}
merge([X|Xs], Ys, Zs) \leftarrow |Zs = [X|Zs'], merge(Xs, Ys, Zs').\\
merge(Xs, [Y|Ys], Zs) \leftarrow |Zs = [Y|Zs'], merge(Xs, Ys, Zs').\\
merge([\,], Ys, Zs) \leftarrow |Ys = Zs.\\
merge(Xs, [\,], Zs) \leftarrow |Xs = Zs.
\end{array}
$$

Se tiene que:

$$
\begin{array}{ll}
\langle merge([a],[b],Zs) \mid \{\}\rangle & \Rightarrow_{C2} \langle merge([],[b],Zs') \mid Zs = [a|Zs']\rangle\\
& \Rightarrow_{C2} \langle |Zs = [a|Zs'], Zs' = [b]\rangle\\
& \Rightarrow \langle \| \boxed{Zs = [a,b]} \rangle\\
& \\
& Zs = [a,b] \vdash_{\mathcal{H},\mathcal{P}^*} merge([a],[b],Zs)
\end{array}
$$

pero no hay garantía de que se elija calcular esta mezcla en vez de $Zs = [b,a]$*, por lo que tenemos un programa incompleto.*

2.4 Programación Concurrente con Restricciones

En esta sección vamos a extender la programación con restricciones ya presentada en la Sección 2.2 añadiendo concurrencia mediante el modelo de cómputo alternativo *CCP*. Expondremos también su sistema de transiciones $\mathcal{T}$, que será la base fundamental para el cálculo propuesto en este trabajo en el Capítulo 4.

2.4.1 El Modelo de Cómputo *CCP*

La *Programación Concurrente con Restricciones* [70, 71] (*CCP*, del inglés *Concurrent Constraint Programming*), usa memoria compartida para los procesos del

programa como un almacén de restricciones, que ha de crecer monótonamente durante el cómputo.

Cada subobjetivo tiene el papel de un *agente* (o proceso), y tiene acceso al almacén de restricciones mediante el uso de las primitivas especiales *ask* y *tell*. La primitiva *ask* detecta cuándo se dan las condiciones adecuadas, consulta el almacén de restricciones para verlo y es bloqueante, para garantizar la sincronización. La primitiva *tell* se encarga de propagar las restricciones a los demás procesos, llevándola a la zona de memoria compartida (*tell* puede ser atómico o eventual).

El modelo de cómputo *CCP* hereda los mecanismos de comunicación por medio de variables lógicas de los lenguajes lógicos concurrentes. De los lenguajes *CLP* hereda los sistemas de restricciones y la sincronización declarativa. Además, su modularidad y su semántica denotacional son típicas de las *álgebras de procesos* [20, 31].

Ejemplo 23. *Con el siguiente programa trataremos de ilustrar la propagación local de restricciones booleanas, usando cláusulas CCP, cuya sintaxis se verá con más detenimiento en la Subsección 2.4.2.*

$$
\begin{array}{l}
and(X,Y,Z) \leftarrow ask\ X = 0 : tell\ Z = 0\ |. \\
and(X,Y,Z) \leftarrow ask\ Y = 0 : tell\ Z = 0\ |. \\
and(X,Y,Z) \leftarrow ask\ Z = 1 : tell\ (X = 1, Y = 1)\ |. \\
and(X,Y,Z) \leftarrow ask\ X = 1 : tell\ Z = Y\ |. \\
and(X,Y,Z) \leftarrow ask\ Y = 1 : tell\ Z = X\ |.
\end{array}
$$

$$
\begin{array}{l}
or(X,Y,Z) \leftarrow ask\ X = 1 : tell\ Z = 1\ |. \\
or(X,Y,Z) \leftarrow ask\ Y = 1 : tell\ Z = 1\ |. \\
or(X,Y,Z) \leftarrow ask\ X = 0 \wedge Y = 0 : tell\ Z = 0\ |. \\
or(X,Y,Z) \leftarrow ask\ Z = 0 : tell\ X = 0 \wedge Y = 0\ |.
\end{array}
$$

$$
\begin{array}{l}
xor(X,Y,Z) \leftarrow ask\ X = 0 \wedge Y = 1 : tell\ Z = 1\ |. \\
xor(X,Y,Z) \leftarrow ask\ X = 1 \wedge Y = 0 : tell\ Z = 1\ |. \\
xor(X,Y,Z) \leftarrow ask\ X = 0 \wedge Y = 0 : tell\ Z = 0\ |. \\
xor(X,Y,Z) \leftarrow ask\ X = 1 \wedge Y = 1 : tell\ X = 0\ |. \\
xor(X,Y,Z) \leftarrow ask\ Z = 0 : tell\ X = Y\ |.
\end{array}
$$

$$
\begin{array}{l}
not(X,Y) \leftarrow ask\ X = 0 : tell\ Y = 1\ |. \\
not(X,Y) \leftarrow ask\ X = 1 : tell\ Y = 0\ |. \\
not(X,Y) \leftarrow ask\ Y = 0 : tell\ X = 1\ |. \\
not(X,Y) \leftarrow ask\ Y = 1 : tell\ X = 0\ |.
\end{array}
$$

Las cláusulas anteriores corresponden a las definiciones en CCP de las funciones

booleanas "and", "or", "xor" y "not", donde los dos primeros argumentos corresponden a los dos argumentos de la función, mientras que el tercer argumento corresponde al resultado. Al programa le añadimos la siguiente cláusula:

$$full\_adder(X,Y,Cin,S,Cout) \leftarrow and(X,Y,C1), xor(X,Y,S1), \\ and(Cin,S1,C2), xor(Cin,S1,S1), \\ or(C1,C2,Cout).$$

la cual corresponde a un sumador de dos números de un bit, X e Y, que representan a los números que se van a sumar, "Cin" es el acarreo de entrada, "S" es el resultado, y "$Cout$" es el acarreo de salida.

Si lanzamos el objetivo $\leftarrow full\_adder(X,Y,1,S,0)$, obtenemos como resultado las restricciones $X = 0, Y = 0, S = 1$, aplicando para ello los siguientes pasos:

1. $or(C1,C2,Cout), Cout = 0 \Rightarrow C1 = 0, C2 = 0.$
2. $and(Cin,S1,C2), C2 = 0, Cin = 1 \Rightarrow S1 = 0.$
3. $xor(X,Y,S1), S1 = 0 \Rightarrow X = Y.$
4. $and(X,Y,C1), X = Y, C1 = 0 \Rightarrow X = 0, Y = 0.$
5. $xor(Cin,S1,S), Cin = 1, S1 = 0 \Rightarrow S = 1.$

2.4.2 *CCP* con Cláusulas Guardadas

El modelo de cómputo *CCP* con *cláusulas guardadas* [68], denominado abreviadamente como $CC(\mathcal{C})$, está formado por cláusulas con la siguiente estructura:

$$H \leftarrow ask(c) : tell(d) \mid \bar{B}.$$

donde H es la cabeza de la cláusula, $ask(c)$ y $tell(d)$ son las guardas que chequean y propagan restricciones, y $\bar{B}$ es el cuerpo de la cláusula, que está formado por una conjunción de átomos.

Definimos una *configuración de estado de cómputo* para *CCP* con cláusulas guardadas de forma similar a los estados de los programas *ALPS*:

$$\langle \leftarrow \bar{A} \mid u \rangle$$

donde $\bar{A}$ es una red de procesos o una conjunción de átomos, y u es el almacén finito de restricciones. Tenemos que una cláusula de la forma:

$$H \leftarrow ask(c) : tell(d) \mid \bar{B}$$

C2	$\langle \leftarrow A \mid u\rangle \Rightarrow \langle \leftarrow \bar{B} \mid u, A = H, c, d\rangle$ si $H \leftarrow ask(c) : tell(d) \mid \bar{B}$ es candidata para A bajo u.
C3	$\dfrac{\langle \leftarrow \bar{A} \mid u\rangle \Rightarrow \langle \leftarrow \bar{A}' \mid u'\rangle}{\langle \leftarrow \bar{B}, \bar{A}, \bar{C} \mid u\rangle \Rightarrow \langle \leftarrow \bar{A}' \mid u'\rangle}$

Figura 2.7: Reglas de computo para *CCP* con cláusulas guardadas.

es *candidata* para un átomo A bajo un almacén de restricciones u si y solo si:

a) $u \vdash_{\mathcal{C}} \exists \bar{x}.(A = H \wedge c)$, estando c validado, y siendo $\bar{x}$ el conjunto de variables locales de c.

b) $\vdash_{\mathcal{C}} \exists \bar{x}.(\bigwedge u \ \wedge A = H \wedge c \wedge d)$, estando d satisfecho, y siendo $\bar{x}$ el conjunto de todas las variables locales.

En particular, se observa que b) expresa que la guarda *tell* es atómica, ya que u representa el almacén global.

Para darle una semántica operacional a *CCP* con cláusulas guardadas, es necesario tener unas reglas de cómputo como las que teníamos en *ALPS*, reglas de fallo para detectar cuándo el cómputo ha acabado sin éxito, y reglas de bloqueo, en el caso de que un átomo o proceso no pueda continuar a la espera de datos. Estas reglas vienen dadas en las tablas de las Figuras 2.7, 2.8 y 2.9, respectivamente.

Ejemplo 24. *Conservando la expresividad del lenguaje, es posible programar el proceso del Ejemplo 22 "merge" en CCP($\mathcal{H}$).*

$$
\begin{array}{l}
merge(Xs, Ys, Zs) \leftarrow \\
\quad ask(\exists X, Xs'.Xs = [X|Xs']) : \\
\quad tell(Xs = [X|Xs'], Zs = [X|Zs'] \mid \\
\quad merge(Xs', Ys, Zs').\\
merge(Xs, Ys, Zs) \leftarrow \\
\quad ask(\exists Y, Ys'.Ys = [Y|Ys']) : \\
\quad tell(Ys = [Y|Ys'], Zs = [Y|Zs'] \mid \\
\quad merge(Xs, Ys', Zs').\\
merge(Xs, Ys, Zs) \leftarrow \\
\quad ask(Xs = [\,]) : tell(Ys = Zs) \mid .
\end{array}
$$

F2	$\langle \leftarrow A \mid u\rangle \Rightarrow \langle \mathit{ff} \mid u\rangle$ si no existe $H \leftarrow ask(c) : tell(d)\vert\bar{B}$ tal que $\vdash_{\mathcal{C}} \exists\bar{x}.(\bigwedge u \ \wedge A = H \wedge c \wedge d)$.
F3	$\dfrac{\langle \leftarrow \bar{A} \mid u\rangle \Rightarrow \langle \leftarrow \mathit{ff} \mid u\rangle}{\langle \leftarrow \bar{B},\bar{A} \mid u\rangle \Rightarrow \langle \leftarrow \mathit{ff} \mid u\rangle}$
F3'	$\dfrac{\langle \leftarrow \bar{A} \mid u\rangle \Rightarrow \langle \leftarrow \mathit{ff} \mid u\rangle}{\langle \leftarrow \bar{A},\bar{B} \mid u\rangle \Rightarrow \langle \leftarrow \mathit{ff} \mid u\rangle}$

Figura 2.8: Reglas de fallo *ff* para *CCP* con cláusulas guardadas.

S2	$\langle \leftarrow A \mid u\rangle \Rightarrow \langle dd \mid u\rangle$ si no son aplicables ni **C2** ni **F2**.
S3	$\dfrac{\langle \leftarrow \bar{A} \mid u\rangle \Rightarrow \langle \leftarrow dd \mid u\rangle \quad \langle \leftarrow \bar{B} \mid u\rangle \Rightarrow \langle \leftarrow dd \mid u\rangle}{\langle \leftarrow \bar{A},\bar{B} \mid u\rangle \Rightarrow \langle \leftarrow dd \mid u\rangle}$

Figura 2.9: Reglas de bloqueo *dd* para *CCP* con cláusulas guardadas.

$$merge(Xs, Ys, Zs) \leftarrow$$
$$\quad ask(Ys = [\,]) : tell(Xs = Zs) \mid .$$

El proceso *append* programado en $ALPS(\mathcal{H})$ se puede programar también en $CCP(\mathcal{H})$, pero al no haber una regla como la **C2a** de *ALPS*, se tiene que:

$$\langle \leftarrow append(Xs, Ys, [\,])\vert\{\}\rangle \not\Rightarrow^* \langle\{\}\vert Xs = [\,], Ys = [\,]\rangle$$

2.4.3 *CCP* con Sintaxis Algebraica

La *Programación Concurrente con Restricciones y Sintaxis Algebraica* [70, 71] (abreviadamente $CC(\mathcal{C})$, siendo $\mathcal{C} = (T, L, \vdash)$ un sistema de restricciones), tiene su sintáxis descrita en notación BNF, para así obtener su lenguaje formal.

A continuación damos el álgebra de este lenguaje $CC(\mathcal{C})$. Primero definiremos la clase g, correspondiente a las primitivas *ask* y *tell*, que consultan y modifican el almacén de restricciones, respectivamente:

$$g ::= ask(c) \mid tell(c) \text{ con } c \in L$$

En $CC(\mathcal{C})$ se sustituyen los objetivos por los agentes A. La definición de *agente* es todo aquello que realiza una acción; en nuestro caso, son los que realizan las acciones del programa.

$$A ::= Stop \mid \sum_{i=1}^{n} g_i \rightarrow A_i \mid A \wedge A \mid \exists x.A \mid p(\bar{x})$$

donde *Stop* es la instrucción de parada, e indica que ya no quedan más instrucciones por ejecutar, el sumatorio indica que hay n alternativas posibles por ejecutar, $\wedge$ indica la composición paralela de dos acciones que deben ejecutarse, el existencial sirve para obtener variables ocultas, y por último, la llamada al procedimiento "$p(\bar{x})$", que ha de estar previamente declarado en D, el conjunto de declaraciones del programa. Las declaraciones de procedimientos son de la siguiente forma.

$$D ::= \epsilon \mid p(\bar{y}) :\!- A, D \text{ siendo } \bar{y} \text{ lineal y } \mathcal{FV}(A) = \bar{y}.$$

Los programas en este lenguaje serán una secuencia de declaraciones de procedimientos, seguidos de un agente inicial que realiza la primera acción:

$$P ::= D.A$$

De forma similar a los paradigmas anteriores, en $CC(\mathcal{C})$ las configuraciones de los estados vienen dadas por la siguiente tupla:

$$\langle\, \alpha \,|\, d \,\rangle$$

donde d es un almacén de restricciones, es decir, una conjunción de elementos de L, y α puede ser un agente A, una guarda, o un modo de terminación $r \in \{ss, ff, dd\}$. Dada una configuración arbitraria como la anterior, se define:

- d satisface $ask(c)$ si y sólo si $d \vdash c$.
- d satisface $tell(c)$ si y sólo si $\vdash \exists \bar{x}.(d \wedge c)$ siendo $\bar{x}$ las variables libres de $d \wedge c$.
- El agente $\sum_{i=1}^{n} g_i \rightarrow A_i$ escoge con compromiso una sola de las guardas g_i satisfechas por d, y continúa ahora con A_i, descartando las demás posibilidades. Falla en el caso de que no haya ninguna guarda g_i satisfecha por d, y en cualquier otro caso se queda en estado de bloqueo.

Por convenio, abreviaremos $tell(c) \rightarrow Stop$ como $tell(c)$. En el siguiente ejemplo se puede ver un programa construido con la sintáxis algebraica que acabamos de especificar.

Ejemplo 25. *Vamos a ver como se programaría el proceso "merge" visto anteriormente en el Ejemplo 24, escrito ahora en CCP con cláusulas guardadas:*

$$
\begin{array}{l}
merge(Xs, Ys, Zs) :- \\
\quad ask(\exists X, Xs'.Xs = [X|Xs']) \\
\quad \rightarrow \exists X, Xs', Zs'.(tell(Xs = [X|Xs']) \wedge \\
\qquad\qquad\qquad\qquad tell(Zs = [X|Zs']) \wedge \\
\qquad\qquad\qquad\qquad merge(Xs', Ys, Zs')) \\
+ \;\; ask(\exists Y, Ys'.Ys = [Y|Ys']) \\
\quad \rightarrow \exists Y, Ys', Zs'.(tell(Ys = [Y|Ys']) \wedge \\
\qquad\qquad\qquad\qquad tell(Zs = [X|Zs']) \wedge \\
\qquad\qquad\qquad\qquad merge(Xs, Ys', Zs')) \\
+ \;\; ask(Xs = []) \rightarrow tell(Zs = Ys) \\
+ \;\; ask(Ys = []) \rightarrow tell(Zs = Xs)
\end{array}
$$

Observamos que, a diferencia de la versión del "merge" para cláusulas guardadas, la gestión del operador existencial $\exists$ se hace explícito en el uso de la primitiva "tell".

Las configuraciones del estado de cómputo pasan de un estado a otro según las *reglas de transición* $\mathcal{T}$. Las reglas para este sistema vienen dadas en las tablas de las Figuras 2.10, 2.11 y 2.12.

En la Figura 2.10 se muestran las reglas para las acciones básicas, como las guardas, que son análogas a CCP con cláusulas guardadas, con excepción de la acción *Stop*, que devuelve un éxito siempre.

La tabla de la Figura 2.11 muestra los pasos de cómputo para la acción de elección con compromiso, cuyo comportamiento definimos en el Ejemplo 23. Las reglas de la Figura 2.12 son las que permiten tratar la concurrencia: la regla **R10** para *interleaving* y las reglas **R11** y **R12** para situaciones de fallo y de bloqueo. Observamos que en la regla **R12** se necesita contemplar la situación de éxito ss para el caso en el que $A \equiv B \equiv Stop$.

A continuación se muestra el ejemplo del sumador con acarreo para ilustrar algunas secuencias de cómputo usando las reglas del sistema de transiciones $\mathcal{T}$.

Ejemplo 26 (**Propagación de restricciones).** *El siguiente programa representa el sumador de un bit con acarreo explicado en el Ejemplo 23 escrito en CCP haciendo uso de la sintaxis algebraica:*

R1	$\langle Stop \mid c\rangle \Rightarrow \langle ss \mid c\rangle$
R2	$\langle tell(c) \mid d\rangle \Rightarrow \langle ss \mid d \wedge c\rangle$ si $\vdash \exists \bar{x}.(d \wedge c)$ (implica consistencia, es decir, $d \wedge c \nvdash F$).
R3	$\langle tell(c) \mid d\rangle \Rightarrow \langle ff \mid d\rangle$ si $d \wedge c \vdash F$(implica que $\nvdash \exists \bar{x}.d \wedge c$).
R4	$\langle ask(c) \mid d\rangle \Rightarrow \langle ss \mid d\rangle$ si $d \vdash c$.
R5	$\langle ask(c) \mid d\rangle \Rightarrow \langle dd \mid d\rangle$ si $d \nvdash c$ pero $\vdash \exists \bar{x}.d \wedge c$.
R6	$\langle ask(c) \mid d\rangle \Rightarrow \langle ff \mid d\rangle$ si $d \wedge c \vdash F$, lo que implica $\nvdash \exists \bar{x}.d \wedge c$.

Figura 2.10: Reglas para acciones básicas del sistema de transiciones $\mathcal{T}$.

$$
\begin{aligned}
and(X,Y,Z) \text{ :- } & ask(X=0) \rightarrow tell(Z=0) \\
+\ & ask(Y=0) \rightarrow tell(Z=0) \\
+\ & ask(Z=1) \rightarrow tell(X=1 \wedge Y=1) \\
+\ & ask(X=1) \rightarrow tell(Y=Z) \\
+\ & ask(Y=1) \rightarrow tell(Y=Z) \\
+\ & ask(X=Y \wedge Z=0) \rightarrow tell(X=0 \wedge Y=0).
\end{aligned}
$$

$$
or(X,Y,Z) \text{ :- } ask(X=1) \rightarrow tell(Z=1)
$$

R7	$\frac{\langle g_i \mid c\rangle \Rightarrow \langle ss \mid d\rangle}{\langle \sum_{j=1}^{n} g_j \rightarrow A_j \mid c\rangle \Rightarrow \langle A_i \mid d\rangle} \ (1 \leq i \leq n)$
R8	$\frac{\langle g_n \mid c\rangle \Rightarrow \langle ff \mid c\rangle \quad \langle \sum_{j=1}^{n-1} g_j \rightarrow A_j \mid c\rangle \Rightarrow \langle r \mid c\rangle}{\langle \sum_{j=1}^{n} g_j \rightarrow A_j \mid c\rangle \Rightarrow \langle r \mid c\rangle}$ siendo $r \in \{ff, dd\}$.
R9	$\frac{\langle g_n \mid c\rangle \Rightarrow \langle dd \mid c\rangle \quad \langle \sum_{j=1}^{n-1} g_j \rightarrow A_j \mid c\rangle \Rightarrow \langle r \mid c\rangle}{\langle \sum_{j=1}^{n} g_j \rightarrow A_j \mid c\rangle \Rightarrow \langle dd \mid c\rangle}$ siendo $r \in \{ff, dd\}$.

Figura 2.11: Reglas para la elección guardada del sistema de transiciones $\mathcal{T}$.

R10	$\frac{\langle A \mid c\rangle \Rightarrow \langle A' \mid d\rangle}{\langle A \wedge B \mid c\rangle \Rightarrow \langle A' \wedge B \mid d\rangle}$
R10'	$\frac{\langle A \mid c\rangle \Rightarrow \langle A' \mid d\rangle}{\langle B \wedge A \mid c\rangle \Rightarrow \langle B \wedge A' \mid d\rangle}$
R11	$\frac{\langle A \mid c\rangle \Rightarrow \langle ff \mid c\rangle}{\langle A \wedge B \mid c\rangle \Rightarrow \langle ff \mid c\rangle}$
R11'	$\frac{\langle A \mid c\rangle \Rightarrow \langle ff \mid c\rangle}{\langle B \wedge A \mid c\rangle \Rightarrow \langle ff \mid c\rangle}$
R12	$\frac{\langle A \mid c\rangle \Rightarrow \langle r \mid c\rangle \quad \langle B \mid c\rangle \Rightarrow \langle r \mid c\rangle}{\langle A \wedge B \mid c\rangle \Rightarrow \langle r \mid c\rangle}$ siendo $r \in \{ss, dd, ff\}$.

Figura 2.12: Reglas para la composición paralela del sistema de transiciones $\mathcal{T}$.

R13	$\langle p(\bar{x}) \mid c\rangle \Rightarrow \langle A[\bar{y}/\bar{x}] \mid c\rangle$ siendo $p(\bar{y})$:- A una declaración de $\mathcal{P}$.

Figura 2.13: Regla para llamadas a procedimientos del sistema de transiciones $\mathcal{T}$.

$$
\begin{array}{l}
\quad + \ ask(Y = 1) \rightarrow tell(Z = 1) \\
\quad + \ ask(X = 0 \wedge Y = 0) \rightarrow tell(Z = 0) \\
\quad + \ ask(Z = 0) \rightarrow tell(X = 0 \wedge Y = 0).
\end{array}
$$

$$
\begin{array}{l}
xor(X, Y, Z) \text{:-} ask(X = 0 \wedge Y = 1) \rightarrow tell(Z = 1) \\
\quad + \ ask(X = 1 \wedge Y = 0) \rightarrow tell(Z = 1) \\
\quad + \ ask(X = 0 \wedge Y = 0) \rightarrow tell(Z = 0) \\
\quad + \ ask(X = 1 \wedge Y = 1) \rightarrow tell(X = 0) \\
\quad + \ ask(Z = 0) \rightarrow tell(X = Y).
\end{array}
$$

$$
\begin{array}{l}
not(X, Y) \text{:-} ask(X = 0) \rightarrow tell(Y = 1) \\
\quad + \ ask(X = 1) \rightarrow tell(Y = 0) \\
\quad + \ ask(Y = 0) \rightarrow tell(X = 1) \\
\quad + \ ask(Y = 1) \rightarrow tell(X = 0).
\end{array}
$$

A continuación damos la siguiente definición de un sumador de un bit con acarreo:

$$
\begin{array}{l}
full\_adder(X, Y, Cin, S, Cout) \text{:-} \\
\quad and(X, Y, C1) \wedge xor(X, Y, S1) \wedge \\
\quad and(Cin, S1, C2) \wedge xor(Cin, S1, S) \wedge \\
\quad or(C1, C2, Cout).
\end{array}
$$

y mostramos dos configuraciones a las que se han aplicado posibles pasos de cómputo, para ilustrar su funcionamiento:

$$
\begin{array}{l}
\langle full\_adder(X, Y, Cin, S, Cout) \wedge tell(Cin = 1) \wedge tell(Cout = 0) | \{\}\rangle \\
\Rightarrow_{R10,R2} \langle full\_adder(X, Y, Cin, S, \overline{Cout) \wedge tell(Cout = 0)} | Cin = 1\rangle \\
\Rightarrow_{R10,R2} \langle \underline{full\_adder(X, Y, Cin, S, Cout)} | \overline{Cin = 1 \wedge Cout = 0}\rangle \\
\Rightarrow_{R13} \langle \overline{and(X, Y, C1) \wedge xor(X, Y, S1) \wedge and(Cin, S1, C2) \wedge xor(Cin, S1, S) \wedge} \\
\overline{or(C1, C2, Cout)} | Cin = 1 \wedge Cout = 0\rangle \\
\Rightarrow_{R10,R13} \langle \overline{and(X}, Y, C1) \wedge xor(X, Y, S1) \wedge and(Cin, S1, C2) \wedge xor(Cin, S1, S) \wedge \\
\qquad (\quad ask(C1 = 1) \rightarrow tell(Cout = 1) \\
\qquad\quad + \ ask(C2 = 1) \rightarrow tell(Cout = 1)
\end{array}
$$

$$
\begin{array}{l}
\quad + \ ask(C1 = 0 \wedge C2 = 0) \rightarrow tell(Z = 0) \\
\quad + \ ask(Cout = 0) \rightarrow \underline{tell(C1 = 0 \wedge C2 = 0)} \quad) \\
|Cin = 1 \wedge Cout = 0\rangle \\
\Rightarrow_{R10,R7} \langle and(X, Y, C1) \wedge xor(X, Y, S1) \wedge and(Cin, S1, C2) \wedge xor(Cin, S1, S) \wedge \\
\underline{tell(C1 = 0 \wedge C2 = 0)}|Cin = 1 \wedge Cout = 0\rangle \\
\Rightarrow_{R10,R2} \langle \underline{and(X, Y, C1)} \wedge xor(X, Y, S1) \wedge \underline{and(Cin, S1, C2)} \wedge xor(Cin, S1, S) \\
|Cin = 1 \wedge Cout = 0 \wedge C1 = 0 \wedge C2 = 0\rangle \\
\Rightarrow_{R10,R13}\Rightarrow_{R10,R7} \langle and(X, Y, C1) \wedge xor(X, Y, S1) \wedge \underline{tell(S1 = C2)} \wedge xor(Cin, S1, S) \\
|Cin = 1 \wedge Cout = 0 \wedge C1 = 0 \wedge C2 = 0\rangle \\
\Rightarrow_{R10,R13}\Rightarrow_{R10,R7} \langle and(X, Y, C1) \wedge xor(X, Y, S1) \wedge \underline{tell(S = 1)}|Cin = 1 \wedge Cout = 0 \wedge \\
C1 = 0 \wedge C2 = 0 \wedge S1 = C2(S1 = 0)\rangle \\
\Rightarrow_{R10,R2} \langle and(X, Y, C1) \wedge \underline{xor(X, Y, S1)}|S = 1 \wedge Cin = 1 \wedge \\
Cout = 0 \wedge C1 = 0 \wedge C2 = 0 \wedge S1 = C2(S1 = 0)\rangle \\
\Rightarrow_{R10,R13}\Rightarrow_{R10,R7} \langle and(X, Y, C1) \wedge \underline{tell(X = Y)}|S = 1 \wedge Cin = 1 \wedge Cout = 0 \wedge \\
C1 = 0 \wedge C2 = 0 \wedge S1 = C2(S1 = 0)\rangle \\
\Rightarrow_{R10,R2} \langle \underline{and(X, Y, C1)}|S = 1 \wedge Cin = 1 \wedge Cout = 0 \wedge C1 = 0 \wedge C2 = 0 \wedge \\
S1 = 0 \wedge X = Y\rangle \\
\Rightarrow_{R13}\Rightarrow_{R7} \langle \underline{tell(X = 0 \wedge Y = 0)}|S = 1 \wedge Cin = 1 \wedge Cout = 0 \wedge C1 = 0 \wedge \\
C2 = 0 \wedge S1 = 0 \wedge X = Y\rangle \\
\Rightarrow_{R2} \langle ss| \ \boxed{S = 1 \wedge X = 0 \wedge Y = 0 \wedge Cin = 1 \wedge Cout = 0} \ \wedge C1 = 0 \wedge C2 = 0 \wedge S1 = \\
0 \wedge X = Y\rangle
\end{array}
$$

Este cómputo muestra la evaluación del predicado del sumador de un bit con acarreo, indicando que el acarreo de entrada "Cin" es 1, y el acarreo de salida "Cout" es 0 mediante el predicado de propagación de restricciones "tell". El cómputo finaliza con éxito y devuelve como respuesta el valor de salida del sumador $S = 1$ y los valores de entrada del sumador $X = 0$ e $Y = 0$. A continuación mostramos dos posibles ramas concurrentes de cómputo para la configuración $\langle not(X, Y) \wedge (tell(X = 0) + tell(Y = 0))|\{\}\rangle$:

$$
\begin{array}{l}
\langle not(X, Y) \wedge (\underline{tell(X = 0)} + tell(Y = 0))|\{\}\rangle \\
\Rightarrow_{R10,R7} \langle not(X, Y) \wedge \underline{tell(X = 0)}|\{\}\rangle \\
\Rightarrow_{R10,R2} \langle \underline{not(X, Y)}|X = 0\rangle \\
\Rightarrow_{R13}\Rightarrow_{R7} \langle \underline{tell(Y = 1)}|X = 0\rangle \\
\Rightarrow_{R2} \boxed{\langle ss|X = 0 \wedge Y = 1\rangle}
\end{array}
$$

En este primer resultado, el cómputo ha escogido propagar la restricción $(X = 0)$, y obtenemos que $Y = 1$ al ser la negación de 0.

$$
\begin{array}{l}
\langle not(X, Y) \wedge (tell(X = 0) + \underline{tell(Y = 0)})|\{\}\rangle \\
\Rightarrow_{R10,R7} \langle not(X, Y) \wedge \underline{tell(Y = 0)}|\{\}\rangle
\end{array}
$$

$\Rightarrow_{R10,R2} \langle \underline{not(X,Y)} | Y = 0 \rangle$
$\Rightarrow_{R13} \Rightarrow_{R7} \langle \underline{tell(X = 1)} | Y = 0 \rangle$
$\Rightarrow_{R2} \boxed{\langle ss | X = 1 \wedge Y = 0 \rangle}$

En el segundo caso, al propagarse primero la restricción $Y = 0$ obtenemos como resultado de la negación que $X = 1$.

2.4.4 Semántica Declarativa de *CCP*

Para definir una semántica declarativa en $CC(\mathcal{C})$, vamos a partir de la conjetura de que para un programa $\mathcal{P}$ se cumple el siguiente resultado de correción:

$$\text{Si } \langle A \mid \top \rangle \Rightarrow^* \langle ss \mid c \rangle \text{ entonces } c \vdash_{\mathcal{P}} A.$$

Esto quiere decir que, si partiendo de una configuración inicial con un objetivo A y la restricción trivial $\top$ se dan una serie de pasos de cómputo según las reglas del sistema de transición $\mathcal{T}$ y llegamos a una configuración final de éxito, entonces se espera que el almacén de restricciones resultante c sea resultado del objetivo A, o bien, c satisface el objetivo A para el programa $\mathcal{P}$. El operador $\vdash_{\mathcal{P}}$ queda definido por las reglas de inferencia dadas en la Figura 2.14.

Como hemos visto anteriormente, los observables almacenan los resultados de los cómputos hasta el final, dando observables de éxito, fracaso, y cómputo infinito. Ahora solo vamos a distinguir entre resultados de cómputo finitos (con resultados de cómputo de éxito y de fracaso) y los resultados de cómputos infinitos:

- $\mathcal{O}(A) =_{def} \{\langle c, \gamma \rangle \mid \langle A \mid \top \rangle \Rightarrow^* \langle \gamma \mid c \rangle\}$ siendo $\gamma \in \{ss, ff, dd\}$ el modo de terminación correspondiente.
- $\mathcal{O}_{ii}(A) =_{def} \{\sqcup_{i \in \mathbb{N}} \bar{c}_i \mid \langle A \mid \top \rangle \Rightarrow \langle A_1 \mid c_1 \rangle \ldots \Rightarrow \langle A_i \mid c_i \rangle \ldots$ es una computación infinita y equitativa$\}$.

Definimos a continuación la *función de comportamiento de entrada/salida* $\mathcal{IO}$ que devuelve el resultado del cómputo del objetivo A, pero en vez de comenzar desde la restricción trivial $\top$, comenzamos desde un conjunto de restricciones c dado:

- $\mathcal{IO}(A) =_{def} \{\langle c, d, \gamma \rangle \mid \langle A \mid c \rangle \Rightarrow^* \langle \gamma \mid d \rangle\}$.
- $\mathcal{IO}_{ii}(A) =_{def} \{\langle c, \sqcup_{i \in \mathbb{N}} \bar{d}_i, \rangle \mid \langle A \mid c \rangle \Rightarrow \langle A_1 \mid d_1 \rangle \ldots \Rightarrow \langle A_i \mid d_i \rangle \ldots$ es una computación infinita y equitativa$\}$.

Semántica Denotacional para *CCP*

Supongamos fijadas las declaraciones de los procedimientos p. Una semántica denotacional para nuestro lenguaje *CCP* es una función ξ de la forma:

Stop	$c \vdash_{\mathcal{P}} Stop$
$\wedge$	$\frac{c \vdash_{\mathcal{P}} A \quad c \vdash_{\mathcal{P}} B}{c \vdash_{\mathcal{P}} A \wedge B}$
$\sum$	$\frac{c \vdash_{\mathcal{P}} g_j \rightarrow A_j}{c \vdash_{\mathcal{P}} \sum_{i=1}^{n} g_i \rightarrow A_i}$
$\rightarrow_{ask}$	$\frac{c \vdash_{\mathcal{C}} d \quad c \vdash_{\mathcal{P}} A}{c \vdash_{\mathcal{P}} ask(d) \rightarrow A}$
$\rightarrow_{tell}$	$\frac{c \vdash_{\mathcal{C}} d \quad c \vdash_{\mathcal{P}} A}{c \vdash_{\mathcal{P}} tell(d) \rightarrow A}$
$\exists$	$\frac{c \vdash_{\mathcal{C}} \exists y.d \quad c \wedge d \vdash_{\mathcal{P}} A[x/y]}{c \vdash_{\mathcal{P}} \exists x.A}$ siendo y una variable fresca.
p	$\frac{c \vdash_{\mathcal{P}} A[\bar{y}/\bar{x}]}{c \vdash_{\mathcal{P}} p(\bar{x})}$ siendo $p(\bar{y})$:- A una declaración de $\mathcal{P}$.

Figura 2.14: Reglas de inferencia para $\vdash_{\mathcal{P}}$.

$$\begin{array}{c} \xi : \text{Agentes} \rightarrow \text{Denotaciones} \\ A \ \mapsto \ \xi[\![A]\!] \end{array}$$

que verifique las propiedades de *composicionalidad* y *corrección* siguientes:

- **Composicionalidad:** La denotación de un agente está determinada por las denotaciones de sus partes:

$$\xi[\![A \, o \, B]\!] = \xi[\![A]\!] \, \tilde{o} \, \xi[\![B]\!]$$

  Tenemos que ξ es un homomorfismo que induce la siguiente congruencia:

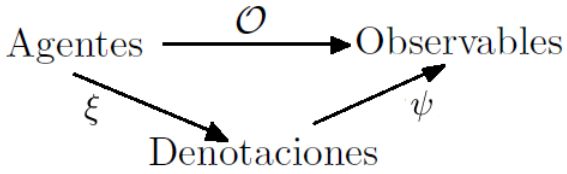

Figura 2.15: Recuperación de observables a partir de las denotaciones de los agentes.

$$\xi[\![A]\!] = \xi[\![A']\!] \text{ y } \xi[\![B]\!] = \xi[\![B']\!] \Rightarrow \xi[\![A \, o \, B]\!] = \xi[\![A' \, o \, B']\!]$$

- **Corrección:** ξ permite recuperar los observables. Al ser un homomorfismo de agentes a denotaciones, debe existir una función ψ que vaya de las denotaciones de los agentes a los observables, como se muestra en la Figura 2.15.

Queremos entonces que ξ sea una abstracción de los observables. Una semántica denotacional ξ se llama *plenamente abstracta* si verifica la siguiente propiedad para todo contexto C:

$$\xi[\![A]\!] = \xi[\![B]\!] \Leftrightarrow \mathcal{O}(C[A]) = \mathcal{O}(C[B])$$

donde:

$\Rightarrow$) Es consecuencia de la composicionalidad y la corrección.

$\Leftarrow$) Sólo se cumple si ξ no discrimina en exceso, es decir, que no introduce información redundante.

La utilidad de la semántica denotacional es dar composicionalidad al lenguaje, lo que es fundamental para poder realizar un *diseño modular*, y para tareas de verificación y análisis [27].

El problema para obtener composicionalidad es que las funciones $\mathcal{O}$ y $\mathcal{IO}$ no son composicionales, como se muestra en los siguientes ejemplos [17, 29]:

Ejemplo 27 (de Boer & Palamidessi, 1994). *Supongamos que tenemos dos conjuntos de restricciones $c, d \in L$ tales que $c \not\vdash_{\mathcal{C}} d$, y los siguientes agentes:*

- $A_1 \equiv ask(c) \rightarrow Stop$
- $A_2 \equiv ask(d) \rightarrow Stop$
- $B \equiv tell(c)$

Los observables, tanto de A_1 como de A_2, tienen ambos el mismo único elemento, que indica que el cómputo queda bloqueado:

$$\mathcal{O}(A_1) = \mathcal{O}(A_2) = \{\langle \top, dd \rangle\}$$

Si fuera composicional, debería darse que $\mathcal{O}(A_1 \wedge B) = \mathcal{O}(A_2 \wedge B)$, pero tenemos que el elemento $\langle c, ss \rangle$ pertenece a los observables de $A_1 \wedge B$ y no a $A_2 \wedge B$. Esto se debe a que si se ejecuta antes A_1 y se queda bloqueado, entonces B añade la restricción c al almacén, A_1 sale del bloqueo, y como se tiene que $c \vdash_{\mathcal{C}} c$ de forma trivial, da éxito. Sin embargo, en $A_2 \wedge B$, al ejecutarse B y luego A_2 (en otro orden se bloquea), falla la primitiva $ask(d)$ por $c \nvdash_{\mathcal{C}} d$. En consecuencia, queda demostrado que $\mathcal{O}$ no es composicional.

Ejemplo 28 (Falaschi & al., 1993). *Supongamos que tenemos los conjuntos de restricciones $a, b, c \in L$ tales que $c \vdash_{\mathcal{C}} b \vdash_{\mathcal{C}} a$ y $a \nvdash_{\mathcal{C}} b \nvdash_{\mathcal{C}} c$, y definimos los siguientes agentes:*

- $A_1 \equiv ask(T) \rightarrow tell(a)$
  $+ \;\; ask(b) \rightarrow tell(c)$
- $A_2 \equiv ask(T) \rightarrow tell(a)$
  $+ \;\; ask(b) \rightarrow tell(c)$
  $+ \;\; ask(T) \rightarrow tell(a) \rightarrow ask(b) \rightarrow tell(c)$
- $B \equiv ask(a) \rightarrow tell(b)$

Los diferentes cómputos para los agentes definidos vienen expresados en los árboles de la Figura 2.16, dónde $r!$ expresa $tell(r)$ y $r?$ expresa $ask(r)$ para $r \in L$.

Tenemos que las dos primeras ramas de A_2 son iguales a las de A_1, y la última rama de A_2 equivale a la concatenación de las dos primeras. Por ello se cumple lo siguiente:

$$\mathcal{IO}(A_1) = \mathcal{IO}(A_2) =$$
$$\{\langle T, a, ss \rangle, \langle a, a, ss \rangle, \langle b, b, ss \rangle, \langle b, c, ss \rangle, \langle c, c, ss \rangle\}$$

La estructura de ramificación de A_1 es diferente a la de A_2; ésto se revela en el contexto $(\Box \wedge B)$, teniendo que:

$$\langle T, c, ss \rangle \in \mathcal{IO}(A_2 \wedge B) \text{ y } \langle T, c, ss \rangle \notin \mathcal{IO}(A_1 \wedge B).$$

Como se ha visto en los ejemplos anteriores, la elección de un camino u otro de cómputo para un agente está influida directamente por los agentes del entorno. Como consecuencia, el conjunto $\mathcal{S}(A)$, definido como el conjunto de las secuencias

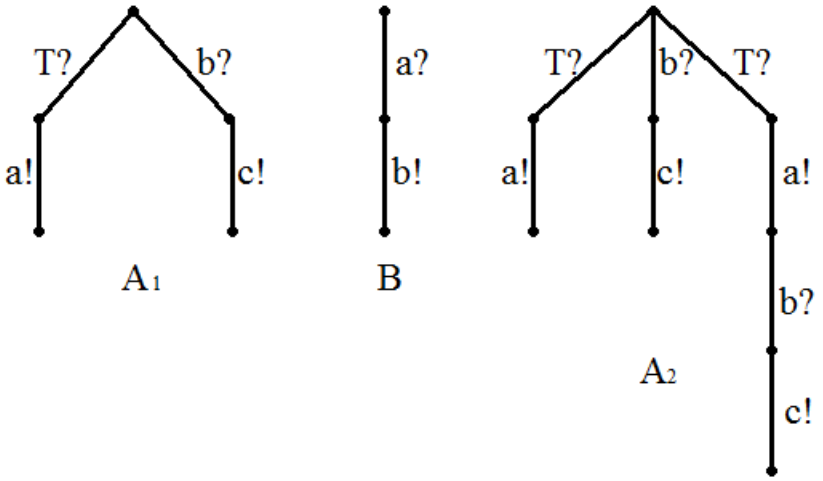

Figura 2.16: Árboles de cómputo para los objetivos A_1, A_2 y B.

de eventos posibles para el agente A, no es composicional.

Semántica Reactiva en *CCP*

La comunicación asíncrona entre agentes (vía el almacén de restricciones) hace que el entorno de un agente tenga menos potencia discriminadora. Ésto implica que es posible una semántica denotacional basada en la observación de secuencias de eventos [18].

Estas secuencias estarán formadas por eventos de dos tipos:

- c^a ($assume$): Supone que el entorno ha enviado la restricción $c \in L$ al almacén.
- c^t ($tell$): Envía la restricción c al almacén.

Una *secuencia reactiva* s formada por los eventos anteriormente descritos tiene la siguiente forma:

$$s \equiv c_1^{l_1}, \ldots, c_n^{l_n} \text{ con } l_i \in \{a, t\}$$
$$cont(s) =_{def} c_1 \wedge \ldots \wedge c_n \ (T \text{ si } n = 0)$$

Con esta definición de secuencias reactivas se puede hallar una función semántica $\mathcal{M}$, que además resulta ser correcta y composicional [18]:

$\mathcal{M}[\![A]\!] =_{def} \{s.\mu | s$ es una secuencia reactiva posible para A, y $\mu \in \{ss, ff, dd, \perp\}\}$

Capítulo 3

Programación Lógico Funcional con Restricciones

En este capítulo se pretende mostrar cómo se han combinado las ventajas de la *programación funcional* con las de la programación lógica con restricciones, dando lugar al esquema de *programación lógico funcional con restricciones* $CFLP(\mathcal{D})$, sobre el cual construiremos un marco concurrente en el capítulo posterior. Comenzamos presentando el paradigma de la *programación funcional*, caracterizada por representar definiciones de funciones como programas, que permite trabajar con elementos de *orden superior* y en la que el valor de las funciones se determina mediante el mecanismo de *evaluación perezosa*, el cual ha demostrado ser un mecanismo de evaluación eficiente basada en *reescritura*. A continuación, introduciremos el esquema $CFLP(\mathcal{D})$ junto con la lógica de reescritura $CRWL(\mathcal{D})$ que proporcionará una semántica declarativa para los $CFLP(\mathcal{D})$-programas. Luego presentaremos la subclase de los $COISS(\mathcal{D})$-programas, que son un subconjunto de los $CFLP(\mathcal{D})$-programas que pueden ser representados mediante *árboles definicionales* y que permiten la evaluación de funciones perezosas no deterministas. Acabaremos el capítulo dando un algoritmo de transformación de $CFLP(\mathcal{D})$-programas a $COISS(\mathcal{D})$-programas que permite generar automáticamente árboles definicionales y que es correcto respecto a la semántica de $CRWL(\mathcal{D})$.

3.1 Programación Funcional

Los *lenguajes de programación funcional* están enraizados en el concepto de *función* (matemática) y su definición mediante *ecuaciones* (generalmente recursivas), que constituyen el programa. Desde el punto de vista computacional, la programación funcional se centra en la evaluación de expresiones (funcionales) para la obtención de un *resultado*.

Ejemplo 29 (**Programación funcional).** *Definimos una función "app" de dos argumentos, que representan las listas que se desea concatenar, y que devuelve como resultado la concatenación de dichas listas. Haciendo uso de la sintaxis del lenguaje funcional* Haskell *[39], en la que el símbolo ':' representa el constructor de listas, el programa consiste en una declaración de tipos, una declaración de la función "app", en la que se fija su dominio y su rango, y dos ecuaciones que la definen:*

$data\ [t] = [\,] \mid (t\ :\ [t])$

$app\ ::\ [t] \to [t] \to [t]$

$app\ [\,]\ xs = xs$
$app\ (x\ :\ xs)\ ys = x : (app\ xs\ ys)$

Un primer rasgo a destacar en el programa del *Ejemplo 29* es la necesidad de emplear recursos expresivos para fijar el perfil de la función, es decir, la naturaleza del dominio y el rango de la función. En programación funcional la descripción de dominios conduce a la idea de *tipo de datos*. Los tipos de datos se introducen en programación para evitar o detectar errores y ayudar a definir estructuras de datos. Los lenguajes funcionales modernos son lenguajes con *disciplina de tipos fuerte* (en inglés, *strongly typed*) que realizan una comprobación de las expresiones que aparecen en el programa para asegurarse de que no se producirán errores durante la ejecución del mismo por incompatibilidades de tipo.

Como ya hemos dicho, un lenguaje funcional se basa en el concepto de función (matemática). Lo que caracteriza a una función (matemática), además de su perfil, es que a cada elemento de su dominio le corresponde un único elemento de su rango; en otras palabras, el resultado (*salida*) de aplicar una función sobre sus argumentos viene determinado exclusivamente por el valor de éstos (su *entrada*). Esta propiedad de las funciones (matemáticas) se denomina *transparencia referencial* y permite el estilo de la programación funcional basado en el razonamiento ecuacional (la sustitución de iguales por iguales) y los cómputos deterministas. Desde el punto de vista computacional, otra propiedad de las funciones (matemáticas) es su capacidad para ser compuestas. La composición de funciones es la técnica por excelencia de la programación funcional, y permite la construcción de programas mediante el empleo de funciones primitivas o previamente definidas por el usuario en sus programas. La composición de funciones refuerza la modularidad de los programas.

Ejemplo 30. *En este ejemplo hacemos uso de la función "app" definida en el* Ejemplo 29 *y la composición de funciones para definir una función que invierte el orden de los elementos de una lista.*

$rev :: [t] \to [t]$

$rev\ [\,]\ = [\,]$
$rev\ (x\ :\ xs)\ = app\ (rev\ xs)\ [x]$

Una de las características de los lenguajes funcionales que va más alla de la mera composición de funciones es el empleo de las mismas como "ciudadanos de primera clase" dentro del lenguaje, de forma que las funciones puedan almacenarse en estructuras de datos, pasarse como argumentos a otras funciones y devolverse como resultados. Agrupamos todas estas características bajo la denominación de *orden superior*.

Ejemplo 31. *Un ejemplo típico de función de orden superior es la función "map", que toma como argumentos una función 'f' y una lista, y forma la lista que resulta de aplicar 'f' a cada uno de los elementos de la lista.*

$map :: (t_1 \to t_2) \to [t_1] \to [t_2]$

$map\ f\ [\,] = [\,]$
$map\ f\ (x\ :\ xs) = (f\ x) : (map\ f\ xs)$

Diversos autores [64, 66] consideran que un programa funcional incluye también, además de una lista de ecuaciones de definición de funciones, una expresión básica (sin variables) que se pretende evaluar como objetivo. La ejecución de un programa consiste entonces en la evaluación de la expresión inicial de acuerdo con las ecuaciones de definición de las funciones y alguna forma de reducción. La *reducción* es un proceso por el cual, mediante una secuencia de pasos, transformamos la expresión inicial, compuesta de símbolos de función y de símbolos de constructoras de datos, en un valor (de su tipo), es decir, una expresión que sólo contiene apariciones de símbolos de constructoras. El valor obtenido se considera el resultado de la evaluación. La secuencia de pasos dada en el proceso de reducción depende de la estrategia de reducción empleada. Podemos distinguir dos *estrategias de reducción* o *modelos de evaluación*, la denominada impaciente y la perezosa. Una estrategia *impaciente* evalúa primero los argumentos de una función mientras que una estrategia *perezosa* evalúa los argumentos sólo si su valor es necesario para el cómputo de dicha función, intentando evitar cálculos innecesarios. Si bien la estrategia impaciente es más fácil de implementar en los computadores con arquitecturas convencionales, puede conducir a secuencias de reducción que no terminan en situaciones en las que una evaluación perezosa es capaz de computar un valor. Este hecho, junto con algunas facilidades de la programación (por ejemplo, la evaluación perezosa libera al programador de la preocupación por el orden de evaluación en las expresiones [38])

y ventajas expresivas que proporcionan (por ejemplo, la habilidad de computar con *estructuras de datos infinitas*), han hecho que la posibilidad de utilizar una estrategia perezosa sea muy apreciada en los lenguajes funcionales modernos.

En cuanto a los formalismos que sustentan la clase de los lenguajes funcionales, se distinguen dos enfoques: el enfoque funcional *clásico* [12, 38, 66] basado en el λ-*cálculo*, y el enfoque *ecuacional* [62, 63] basado en la *lógica ecuacional*. Aunque los dos enfoques difieren en la sintaxis y semántica del formalismo empleado (el enfoque ecuacional se suele restringir a primer orden, aunque es posible extenderlo a orden superior [65] usando también recursos del λ-cálculo [24]), en ambos casos la *reescritura* [11] juega un papel en la reducción de una expresión dada a formal normal o irreducible, que es lo que persigue un cómputo. La utilización de la reescritura permite además establecer una correspondencia sencilla con el formalismo que sustenta la programación lógica.

3.2 El Esquema $CFLP(\mathcal{D})$

En el esta sección introduciremos las principales características del esquema $CFLP(\mathcal{D})$ [52] sobre el que se basará nuestro cálculo concurrente. Comenzaremos explicando aquellos conceptos previos que son necesarios para su comprensión, y pasaremos a explicar la sintaxis de los $CFLP(\mathcal{D})$-programas, junto con la lógica de reescritura $CRWL(\mathcal{D})$ para proporcionar una semántica declarativa a dichos programas.

3.2.1 Expresiones Aplicativas, Patrones y Sustituciones

Vamos a introducir brevemente la sintaxis de las expresiones aplicativas y de los patrones, los cuales son necesarios para entender la construcción posterior de dominios de restricciones y resolutores de restricciones.

Asumiremos una signatura universal $\Sigma = \langle DC, FS \rangle$, donde $DC = \bigcup_{n \in \mathbb{N}} DC^n$ y $FS = \bigcup_{n \in \mathbb{N}} FS^n$ son familias de conjuntos infinitos numerables y mutuamente disjuntos de *constructoras de datos* y de *símbolos de funciones evaluables* respectivamente, cada uno con una aridad asociada. Escribiremos $\Sigma_\perp$ para el resultado de extender DC^0 con el símbolo especial $\perp$, con el objetivo de denotar un valor indefinido para un dato. Por convenio notacional, usamos $c, d \in DC$, $f, g \in FS$ y $h \in DC \cup FS$, y definimos la *aridad* de $h \in DC^n \cup FS^n$ como $ar(h) = n$. También asumimos que DC^0 incluye las constantes $true$, $false$ y $success$, las cuales nos son útiles para representar los resultados devueltos por varias funciones primitivas. Asumiremos a continuación que tenemos un conjunto infinito numerable $\mathcal{V}$ de *variables* $X, Y, \ldots$ y un conjunto $\mathcal{U}$ de *elementos primitivos* $u, v, \ldots$, mutuamente disjuntos, y disjuntos a su vez de $\Sigma_\perp$. Los elementos primitivos están destinados a representar conjuntos de valores de dominio específico, como por ejemplo el conjunto $\mathbb{R}$ de los números reales

usado en el lenguaje $CLP(\mathcal{R})$ [41] o el conjunto $\mathbb{Z}$ de los números enteros usado en el lenguaje $CLP(\mathcal{FD})$ [37].

Además, definimos las *expresiones parciales* $e \in Exp_\perp(\mathcal{U})$ con la siguiente sintaxis:

$$e ::= \perp \mid u \mid X \mid h \mid (e\, e_1)$$

donde $u \in \mathcal{U}$, $X \in \mathcal{V}$, $h \in DC \cup FS$. A esta forma de definir las expresiones se les denomina *aplicativas* porque $(e\, e_1)$ representa la operación de *aplicación* (representada como yuxtaposición), donde se aplica la función denotada por e al argumento denotado por e_1. La sintaxis aplicativa es común en lenguajes funcionales de orden superior. La sintaxis de primer orden para expresiones puede ser traducida a sintaxis aplicativa, también llamada *notación currificada*. Por ejemplo, $f(X, g(Y))$ sería $(f\, X\, (g\, Y))$. Siguiendo los convenios usuales, asumiremos que la aplicación es asociativa por la izquierda, y usaremos la notación $(e\, \overline{e}_n)$ para abreviar $(e\, e_1 \ldots e_n)$.

Denotaremos $var(e)$ como el conjunto de variables que aparecen en e. Una expresión e se dice que es *lineal* si y sólo si no existe ninguna $X \in var(e)$ que tenga más de una aparición en e. Debido a su utilidad, también tendremos en cuenta la siquiente clasificación de expresiones: $(X\, \overline{e}_m)$, con $X \in \mathcal{V}$ y $m \geq 0$, se dice que es una *expresión flexible*, mientras que $u \in \mathcal{U}$ y $(h\, \overline{e}_m)$ con $h \in DC \cup FS$ son llamadas *expresiones rígidas*. Además, una expresión rígida $(h\, \overline{e}_m)$ se dice que es *activa* si y sólo si $h \in FS$ y $m \geq ar(h)$, y *pasiva* en cualquier otro caso. Intuitivamente se puede ver que reducir una expresión hasta la raíz solo tiene sentido si la expresión es activa. Algunos subconjuntos interesantes de $Exp_\perp(\mathcal{U})$ son: $GExp_\perp(\mathcal{U})$, el conjunto de las expresiones *cerradas* e tales que $var(e) = \emptyset$; $Exp(\mathcal{U})$, el conjunto de las expresiones *totales* e sin ninguna ocurrencia de $\perp$; $GExp(\mathcal{U})$, el conjunto de las expresiones cerradas y totales $GExp_\perp(\mathcal{U}) \cap Exp(\mathcal{U})$.

Otra subclase importante de las expresiones es el conjunto de los *patrones parciales* $s, t \in Pat_\perp(\mathcal{U})$, cuya sintaxis se define a continuación:

$$t ::= \perp \mid u \mid X \mid c\, \overline{t}_m \mid f\, \overline{t}_m$$

donde $u \in \mathcal{U}$, $X \in \mathcal{V}$, $c \in DC^n$, $m \leq n$, $f \in FS^n$, $m < n$. Nótese que las expresiones $(f\, \overline{t}_m)$ con $f \in FS^n$, $m \geq n$ no están permitidas como patrones, porque son potencialmente evaluables usando una función primitiva o dada una definición de función f. Los patrones de la forma $(f\, \overline{t}_m)$ con $f \in FS^n$, $m < n$, han sido usados en programación lógico funcional [61] como una representación para valores de orden superior. Los subconjuntos $Pat(\mathcal{U})$, $GPat_\perp(\mathcal{U})$, $GPat(\mathcal{U}) \subseteq Pat_\perp(\mathcal{U})$ de patrones *totales*, *cerrados y totales* y *cerrados* son definidos de forma anaáloga.

Siguiendo el espíritu de la semántica denotacional [32], entenderemos $Pat_\perp(\mathcal{U})$ como un conjunto de elementos finitos de un dominio semántico, y definimos el *orden de información* $\sqsubseteq$ como el menor orden parcial sobre $Pat_\perp(\mathcal{U})$ que satisface las siguientes propiedades: $\perp \sqsubseteq t$ para todo $t \in Pat_\perp(\mathcal{U})$, y $(h\bar{t}_m) \sqsubseteq (h\overline{t'}_m)$ siempre que esas dos expresiones sean patrones y $t_i \sqsubseteq t'_i$ para todo $1 \leq i \leq m$. De aquí en adelante, $\bar{t}_m \sqsubseteq \overline{t'}_m$ será entendido como el significado de $t_i \sqsubseteq t'_i$ para todo $1 \leq i \leq m$. Nótese que un patrón $t \in Pat_\perp(\mathcal{U})$ es maximal con respecto a la información de ordenación si y sólo si t es un patrón total, es decir, $t \in Pat(\mathcal{U})$. Para algunos propósitos resulta útil extender la información ordenando el conjunto de todas las expresiones parciales. Esta extensión es definida como el mínimo orden parcial sobre $Exp_\perp(\mathcal{U})$ que verifica $\perp \sqsubseteq e$ para todo $e \in Exp_\perp(\mathcal{U})$, y $(e\, e_1) \sqsubseteq (e'\, e'_1)$ siempre que $e \sqsubseteq e'$ y $e_1 \sqsubseteq e'_1$.

Redefiniremos las *sustituciones* $\sigma \in Sub_\perp(\mathcal{U})$ como aplicaciones $\sigma : \mathcal{V} \to Pat_\perp(\mathcal{U})$ extendidas a $\sigma : Exp_\perp(\mathcal{U}) \to Exp_\perp(\mathcal{U})$ de forma natural. De forma similar, consideramos las *sustituciones totales* $\sigma \in Sub(\mathcal{U})$ dadas por la aplicación $\sigma : \mathcal{V} \to Pat(\mathcal{U})$, *sustituciones cerradas* $\sigma \in GSub_\perp(\mathcal{U})$ dadas por las aplicaciones $\sigma : \mathcal{V} \to GPat_\perp(\mathcal{U})$, y las *sustituciones totales cerradas* $\sigma \in GSub(\mathcal{U})$ dadas por las aplicaciones $\sigma : \mathcal{V} \to GPat(\mathcal{U})$. Por convenio, escribimos ε para la sustitución identidad, $e\sigma$ en lugar de $\sigma(e)$, y $\sigma\theta$ para la composición de σ y θ, tal que $e(\sigma\theta) = (e\sigma)\theta$ para cualquier $e \in Exp_\perp(\mathcal{U})$. Definimos los dominios y el rango de variable de una sustición de la siguiente forma: $Dom(\sigma) = \{X \in \mathcal{V} \mid \sigma(X) \neq X\}$ y $ran(\sigma) = \bigcup_{X \in Dom(\sigma)} var(\sigma(X))$. Como de costumbre, una sustitución σ tal que $Dom(\sigma) \cap ran(\sigma) = \emptyset$ se dice que es *idempotente*. Para cualquier conjunto de variables $\mathcal{X} \subseteq \mathcal{V}$ definimos la *restricción* $\sigma \upharpoonright \mathcal{X}$ como la sustitución σ' tal que $Dom(\sigma') = \mathcal{X}$ y $\sigma'(X) = \sigma(X)$ para todo $X \in \mathcal{X}$. Usamos la notación $\sigma =_\mathcal{X} \theta$ para indicar que $\sigma \upharpoonright \mathcal{X} = \theta \upharpoonright \mathcal{X}$, y abreviamos $\sigma =_{\mathcal{V} \setminus \mathcal{X}} \theta$ como $\sigma =_{\setminus \mathcal{X}} \theta$. Finalmente, consideramos dos formas de comparar las ustituciones dadas $\sigma, \sigma' \in Sub_\perp(\mathcal{U})$. σ se dice que es *menos concreto* que σ' sobre $\mathcal{X} \subseteq \mathcal{V}$ (en símbolos, $\sigma \leq_\mathcal{X} \sigma'$) si y sólo si $\sigma\theta =_\mathcal{X} \sigma'$ para algún $\theta \in Sub_\perp(\mathcal{U})$. La notación $\sigma \leq \sigma'$ abrevia $\sigma \leq_\mathcal{V} \sigma'$. Se dice que σ soporta menos información que σ' sobre $\mathcal{X} \subseteq \mathcal{V}$ (en símbolos, $\sigma \sqsubseteq_\mathcal{X} \sigma'$) si y sólo si $\sigma(X) \sqsubseteq \sigma'(X)$ para todo $X \in \mathcal{X}$. La notación $\sigma \sqsubseteq \sigma'$ se usa para abreviar $\sigma \sqsubseteq_\mathcal{V} \sigma'$.

3.2.2 Restricciones sobre un Dominio de Restricciones

Intuitivamente, un *dominio de restricciones* se espera que proporcione un conjunto de elementos primitivos representando datos específicos junto a una serie de funciones primitivas y de predicados que operan sobre ellos. La siguiente definición extiende la noción de dominio de restricciones $\mathcal{D}$ introducida en [52] añadiendo un *resolutor de restricciones*:

Definición 32 (**Dominio de Restricciones**).

1. *Una* signatura de restricciones *es cualquier familia* $PF = \bigcup_{n\in\mathbb{N}} PF^n$ *de símbolos de funciones primitivas* p*, cada una con una aridad asociada, tal que* $PF^n \subseteq FS^n$ *para cada* $n \in \mathbb{N}$.

2. *Un* dominio de restricciones *de una signatura* PF *es cualquier estructura* $\mathcal{D} = \langle D_\mathcal{U}, \{p^\mathcal{D} \mid p \in PF\}, Solve^\mathcal{D}\rangle$ *tal que el conjunto soporte* $D_\mathcal{U} = GPat_\bot(\mathcal{U})$ *coincide con el conjunto de patrones cerrados para algún conjunto de elementos primitivos* $\mathcal{U}$*, la interpretación* $p^\mathcal{D}$ *de cada* $p \in PF^n$ *satisface los siguientes requisitos:*

   (a) $p^\mathcal{D} \subseteq D_\mathcal{U}^n \times D_\mathcal{U}$*, que se reduce a* $p^\mathcal{D} \subseteq D_\mathcal{U}$ *en el caso de que* $n = 0$*. De aquí en adelante siempre escribiremos* $p^\mathcal{D}\,\bar{t}_n \to t$ *para indicar que* $(\bar{t}_n, t) \in p^\mathcal{D}$*. En el caso* $n = 0$*, esta notación se reduce a* $p^\mathcal{D} \to t$.

   (b) $p^\mathcal{D}$ *se comporta* monótonamente *en sus argumentos y* antimonótonamente *en sus resultados; es decir, si* $p^\mathcal{D}\,\bar{t}_n \to t$, $\bar{t}_n \sqsubseteq \overline{t'}_n$ *y* $t \sqsupseteq t'$ *se tiene entonces* $p^\mathcal{D}\,\overline{t'}_n \to t'$.

   (c) $p^\mathcal{D}$ *se comporta* radicalmente *en el siguiente sentido: siempre que* $p^\mathcal{D}\,\bar{t}_n \to t$ *y* $t \neq \bot$*, hay algún patrón total* $t' \in D_\mathcal{U}$ *tal que* $p^\mathcal{D}\,\bar{t}_n \to t'$ *y* $t' \sqsupseteq t$.

   y $Solve^\mathcal{D}$ *es un* resolutor de restricciones*, cuyo comportamiento será explicado en la* Definición 36.

De ahora en adelante, vamos a suponer un dominio de restricciones arbitrariamente fijado $\mathcal{D}$ construido sobre un cierto conjunto de elementos primitivos $\mathcal{U}$, sobre el cual definiremos la sintaxis de las *restricciones* sobre $\mathcal{D}$. De aquí en adelante, escribiremos $DF = FS \setminus PF$ para el conjunto de símbolos de función definidas por el usuario, y $DF^n = FS^n \setminus PF^n$ para el conjunto de símbolos de función definidos por el usuario de aridad n. La siguiente definición distingue entre restricciones primitivas, sin ninguna aparición activa de símbolos de funciones definidas, y restricciones definidas por el usuario que pueden tener dichas ocurrencias. Para abreviar, en algunas ocasiones escribiremos simplemente 'restricciones' en vez de 'restricciones definidas por el usuario'.

Definición 33 (**Sintaxis de las Restricciones**).

1. *Las* restricciones primitivas atómicas *tienen la forma sintáctica* $p\,\bar{t}_n \to !\, t$*, con* $p \in PF^n$*,* $t_i \in Pat_\bot(\mathcal{U})$ *para todo* $1 \leq i \leq n$*, y* $t \in Pat(\mathcal{U})$*. Las constantes especiales* $\Diamond$ *y* $\blacklozenge$ *también son restricciones primitivas atómicas.*

2. *Las* restricciones primitivas *son construidas usando restricciones primitivas atómicas a través de la conjunción lógica* $\wedge$ *y el cuantificador existencial* $\exists$.

3. *Las* restricciones atómicas *tienen la estructura sintáctica* $p\,\overline{e}_n \to\!!\, t$ *, con* $p \in PF^n$*,* $e_i \in Exp_\perp(\mathcal{U})$ *para todo* $1 \leq i \leq n$*, y* $t \in Pat(\mathcal{U})$*. Las constantes* $\Diamond$ *y* $\blacklozenge$ *también son restricciones atómicas.*

4. *Las* restricciones *son construidas usando restricciones atómicas a través de la conjunción* $\wedge$ *y el cuantificador existencial* $\exists$*.*

De aquí en adelante usaremos las siguientes notaciones: $PCon_\perp(\mathcal{D})$ para el conjunto de todas las restricciones primitivas π sobre $\mathcal{D}$ y $PCon(\mathcal{D})$ para el conjunto de todas las restricciones primitivas totales sobre el dominio $\mathcal{D}$, definido como $\{\pi \in PCon_\perp(\mathcal{D}) \mid \pi \text{ sin apariciones de } \perp\}$. Además, escribimos $DCon_\perp(\mathcal{D})$ para denotar el conjunto de todas las restricciones definidas por el usuario δ sobre $\mathcal{D}$, así como $DCon(\mathcal{D})$ para el subconjunto $DCon_\perp(\mathcal{D})$ de restricciones totales. Reservaremos las mayúsculas Π y C para referirnos a conjuntos de restricciones primitivas o definidas por el usuario respectivamente, y puede interpretarse como la conjunción de todos sus elementos. La semántica de las restricciones primitivas depende de la noción de *solución*, presentada en la siguiente definición.

Definición 34 (**Soluciones de Restricciones Primitivas**). 1. *El conjunto de* valoraciones *y el conjunto de* valoraciones totales *sobre* $\mathcal{D}$ *se definen como* $Val_\perp(\mathcal{D}) = GSub_\perp(\mathcal{U})$ *y* $Val(\mathcal{D}) = GSub(\mathcal{U})$*, respectivamente.*

2. *El conjunto de soluciones de* $\pi \in PCon_\perp(\mathcal{D})$ *es un subconjunto* $Sol_\mathcal{D}(\pi) \subseteq Val_\perp(\mathcal{D})$ *definido de forma recursiva:*

   (a) $Sol_\mathcal{D}(\Diamond) = Val_\perp(\mathcal{D})$ *y* $Sol_\mathcal{D}(\blacklozenge) = \emptyset$*.*

   (b) $Sol_\mathcal{D}(p\,\overline{t}_n \to\!!\, t) = \{\eta \in Val_\perp(\mathcal{D}) \mid t\eta \text{ es total y } p^\mathcal{D}\,\overline{t}_n\eta \to t\eta\}$*.*

   (c) $Sol_\mathcal{D}(\pi_1 \wedge \pi_2) = Sol_\mathcal{D}(\pi_1) \cap Sol_\mathcal{D}(\pi_2)$*.*

   (d) $Sol_\mathcal{D}(\exists X.\pi) = \{\eta \in Val_\perp(\mathcal{D}) \mid \eta' \in Sol_\mathcal{D}(\pi) \text{ para algún } \eta' =_{\setminus\{X\}} \eta\}$*.*

3. *El conjunto de soluciones de* $\Pi \subseteq PCon_\perp(\mathcal{D})$ *está definido como* $Sol_\mathcal{D}(\Pi) = \bigcap_{\pi\in\Pi} Sol_\mathcal{D}(\pi)$*, correspondiendo a la lectura lógica de* Π *como la conjunción de sus partes. Más concretamente,* $Sol_\mathcal{D}(\emptyset) = Val_\perp(\mathcal{D})$*, correspondiendo a la lectura lógica de una conjunción vacía de forma idéntica a la restricción* $\Diamond$*.*

Usando la noción de solución, podemos introducir algunas nociones semánticas útiles relacionadas con las restricciones primitivas:

Definición 35 (**Nociones Primitivas Semánticas**). *Suponemos un conjunto finito* $\Pi \subseteq PCon_\perp(\mathcal{D})$ *de restricciones primitivas, una restricción primitiva* $\pi \in PCon_\perp(\mathcal{D})$*, las expresiones* $e, e' \in Exp_\perp(\mathcal{U})$*, los patrones* $\overline{t}_n$*,* $t \in Pat_\perp(\mathcal{U})$*, y un símbolo de función definida* $p \in PF^n$*, definimos:*

1. π *se dice que es* satisfactible *en* $\mathcal{D}$ *(en símbolos* $Sat_\mathcal{D}(\pi)$*) si y sólo si* $Sol_\mathcal{D}(\pi) \neq \emptyset$*. En cualquier otro caso se dice que* π *es* insatisfactible *(en símbolos* $InSat_\mathcal{D}(\pi)$*). Análogamente para conjuntos de restricciones.*

2. *π es una* consecuencia *de Π en $\mathcal{D}$ (en símbolos, $\Pi \models_{\mathcal{D}} \pi$) si y sólo si $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{D}}(\pi)$. Más concretamente, $p\,\overline{t}_n \to ! t$ es una consecuencia de Π en $\mathcal{D}$ (en símbolos, $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \to ! t$) si y sólo si $p^{\mathcal{D}}\,\overline{t}_n\eta \to t\eta$ con $t\eta$ total para todo $\eta \in Sol_{\mathcal{D}}(\Pi)$.*

3. *$e \sqsubseteq e'$ es una* consecuencia *de Π en $\mathcal{D}$ (en símbolos, $\Pi \models_{\mathcal{D}} e \sqsubseteq e'$) si y sólo si $e\eta \sqsubseteq e'\eta$ se cumple para todo $\eta \in Sol_{\mathcal{D}}(\Pi)$.*

Llegados a este punto, podemos especificar el comportamiento esperado del resolutor de restricciones $Solve^{\mathcal{D}}$ presentado en la *Definición 32*. La siguiente definición está inspirada en [10, 49, 54].

Definición 36 (Resolutores de Restricciones). *1. Diremos que una variable $X \in \mathcal{V}$ es* demandada *por un conjunto de restricciones primitivas $\Pi \subseteq PCon(\mathcal{D})$ si y sólo si $\mu(X) \neq \bot$ se cumple para cada $\mu \in Sol_{\mathcal{D}}(\Pi)$. Escribimos $DVar_{\mathcal{D}}(\Pi)$ para denotar el conjunto de variables demandadas por Π. Para dominios de restricciones prácticos, $DVar_{\mathcal{D}}(\Pi)$ se espera que sea computable (ver Apartado 38).*

2. *Un* resolutor de restricciones *sobre un dominio de restricciones $\mathcal{D}$ es una función denotada por $Solve^{\mathcal{D}}$ que espera como parámetros un conjunto finito $S \subseteq PCon(\mathcal{D})$ de restricciones primitivas atómicas (al que llamamos* almacén de restricciones*) y un conjunto finito de variables $\chi \subseteq \mathcal{V}$ (llamado conjunto de* variables protegidas*). El resolutor se espera que devuelva una disyunción finita $Solve^{\mathcal{D}}(S, \chi) = \bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i)$ que satisfaga los siguientes requisitos:*

   (a) *Cada $S_i \subseteq PCon(\mathcal{D})$ es un conjunto finito de de restricciones primitivas atómicas tales que $Solve^{\mathcal{D}}(S_i, \chi) = S_i \,\square\, \varepsilon$ (es decir, S_i es en χ-*forma resuelta*). Además, $var(S_i) \cap \chi = \emptyset$ si no $DVar_{\mathcal{D}}(S_i) \cap \chi \neq \emptyset$.*

   (b) *Cada $\sigma_i \in Sub(\mathcal{U})$ es una sustitución idempotente de patrones totales para variables tales que $Dom(\sigma_i) \cap var(S_i) = \emptyset$ y $\chi \cap (Dom(\sigma_i) \cup ran(\sigma_i)) = \emptyset$.*

   (c) *$Sol_{\mathcal{D}}(S) = \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus S}.\ S_i \,\square\, \sigma_i)$, donde $\exists_{\setminus S}$ es el cuantificador existencial para todas las variables $\overline{U}_i$ en $S_i \,\square\, \sigma_i$ $(1 \leq i \leq k)$ sin apariciones libres en S y $Sol_{\mathcal{D}}(\exists\overline{U}_i.\ S_i \,\square\, \sigma_i) = \{\mu \in Val_{\bot}(\mathcal{D}) \mid$ existe $\mu' \in Val_{\bot}(\mathcal{D})$ tal que $\mu' =_{\setminus \overline{U}_i} \mu$, $\mu' \in Sol_{\mathcal{D}}(S_i)$ y $X\mu' \equiv t\mu'$ para cada $X \mapsto t \in \sigma_i\}$.*

   En el caso $k = 0$, $\bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i)$ se entiende como $\blacklozenge$. En ese caso, $Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{D}}(\blacklozenge) = \emptyset$ significa la detección de fallo.

Desde un punto de vista operacional, los resolutores ofrecen una elección entre k alternativas. Las alternativas no pueden ligar variables. Además, para cada alternativa $S_i \,\square\, \sigma_i$, o bien las variables protegidas desaparecen, o bien alguna variable

protegida se vuelve demandada.

A continuación vamos a redefinir el sistema de restricciones $\mathcal{H}$ visto en el capítulo anterior con la definición de dominio de restricciones dada en este capítulo y mostraremos un resolutor para dicho dominio.

Ejemplo 37 (**El dominio de restricciones $\mathcal{H}$**). *Consideramos el dominio de restricciones $\mathcal{H}$ construido a partir de un conjunto vacío de elementos primitivos descrito en la Subsección 2.2.3 del capítulo anterior, teniendo la igualdad estricta "seq" como su única primitiva, cuyo comportamiento se interpreta a continuación como: $seq^{\mathcal{H}_{seq}}\, t\, t \to true$ para todo t total $t \in GPat(\emptyset)$; $seq^{\mathcal{H}_{seq}}\, t\, s \to false$ para todo $t, s \in GPat_{\bot}(\emptyset)$ tal que t y s no tienen cota superior con respecto al orden de información '$\sqsubseteq$'; $seq^{\mathcal{H}_{seq}}\, t\, s \to \bot$ en cualquier otro caso. De ahora en adelante, usaremos $t == s$ para abreviar $seq\ t\ s \to! true$, $t /= s$ para abreviar $seq\ t\ s \to! false$ y $Tot(t)$ para abreviar $seq\ t\ t \to! true$.*

Un posible resolutor de restricciones $Solve^{\mathcal{H}_{seq}}$ puede ser encontrado en la siguiente definición.

Definición 38 (**El Resolutor de Restricciones $Solve^{\mathcal{H}_{seq}}$**). *Asumimos el dominio de restricciones $\mathcal{H}_{seq}$ descrito en el* Ejemplo 37. *Sea $\chi \subseteq \mathcal{V}$ un conjunto de variables protegidas y $S \subseteq PCon(\mathcal{H}_{seq})$ una conjunción de restricciones primitivas atómicas sobre $\mathcal{H}_{seq}$ de la forma $S \equiv seq\ t_1\ s_1 \to!\ r_1, \ldots, seq\ t_n\ s_n \to!\ r_n$, donde $t_i, s_i \in Pat(\emptyset)$ y $r_i \in \{true, false\} \cup \mathcal{V}$. Definimos un* resolutor de restricciones *para restricciones de igualdad y de desigualdad tal como se muestra a continuación: $Solve^{\mathcal{H}_{seq}}(S, \chi) = \bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i) \Leftrightarrow_{def} S \,\square\, \varepsilon \rightsquigarrow_{\chi}^{*} \bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i) \not\rightsquigarrow_{\chi}$, donde la relación $\rightsquigarrow_{\chi}$ definida a continuación denota un paso del resolutor de restricciones, y $\varphi \not\rightsquigarrow_{\chi}$ expresa que φ no es reducible por $\rightsquigarrow_{\chi}$.*

El siguiente sistema de reglas especifica el comportamiento de la relación $\rightsquigarrow_{\chi}$ entre disyunciones de restricciones de la forma $\bigvee_i(S_i \,\square\, \sigma_i)$, tal que cada $S_i \subseteq PCon(\mathcal{H}_{seq})$, $\sigma_i \in Sub(\emptyset)$, $\chi \cap var(S_i) = \emptyset$ y $var(S_i) \cap Dom(\sigma_i) = \emptyset$. Cuando aplicamos las reglas de $\rightsquigarrow_{\chi}$, ignoramos el orden de S e interpretamos $==$ y $/=$ como simétricos.

Reglas generales para $\rightsquigarrow_{\chi}$

R0 $\ldots \vee S_i \,\square\, \sigma_i \vee \ldots \rightsquigarrow_{\chi} \ldots \vee \bigvee_j(S'_j \,\square\, \sigma'_j) \vee \ldots$ si $S_i \,\square\, \sigma_i \rightsquigarrow_{\chi} \bigvee_j(S'_j \,\square\, \sigma'_j)$.
R1 $seq\ t\ s \to! R, S \,\square\, \sigma \rightsquigarrow_{\chi} (t == s, S\theta_1 \,\square\, \sigma\theta_1) \vee (t /= s, S\theta_2 \,\square\, \sigma\theta_2)$
si $R \notin \chi$, $\theta_1 = \{R \mapsto true\}$ y $\theta_2 = \{R \mapsto false\}$.

Reglas para la igualdad estricta

R2 $h\bar{t}_n == h\bar{s}_n, S \,\square\, \sigma \rightsquigarrow_{\chi} t_1 == s_1, \ldots, t_n == s_n, S \,\square\, \sigma$

R3 $X == t, S \,\square\, \sigma \rightsquigarrow_\chi Tot(t), S\theta \,\square\, \sigma\theta$
si $X \notin \chi \cup var(t)$, $var(t) \cap \chi = \emptyset$ y $\theta = \{X \mapsto t\}$.

Reglas para la desigualdad estricta

R4 $h\bar{t}_n /= h\bar{s}_n, S \,\square\, \sigma \rightsquigarrow_\chi \bigvee_{i=1}^{n}(t_i /= s_i, S \,\square\, \sigma)$
R5 $h\bar{t}_n /= h'\bar{s}_m, S \,\square\, \sigma \rightsquigarrow_\chi S \,\square\, \sigma$ si $h \neq h'$ o $m \neq n$.
R6 $X /= h\bar{t}_n, S \,\square\, \sigma \rightsquigarrow_\chi (\bigvee_i(S\theta_i \,\square\, \sigma\theta_i)) \vee (\bigvee_{k=1}^{n}(U_k /= t_k\theta, S\theta \,\square\, \sigma\theta))$
si $X \notin \chi$, $var(\bar{t}_n) \cap \chi \neq \emptyset$, $\theta_i = \{X \mapsto h_i\overline{Y}_{in_i}\}$ para cada $h_i \neq h$ y $\theta = \{X \mapsto h\overline{U}_n\}$ con $\overline{Y}_{in_i}, \overline{U}_n$ nuevas variables.

En la práctica, **R6** *se usa para limitar la elección de $h_i\overline{Y}_{in_i}$ a patrones del mismo tipo que X. Así pues, el número de elecciones 'n' es finito.*

Reglas de fallo

R7 $h\bar{t}_n == h'\bar{s}_m, S \,\square\, \sigma \rightsquigarrow_\chi \blacklozenge$ si $h \neq h'$ o $m \neq n$.
R8 $X == t, S \,\square\, \sigma \rightsquigarrow_\chi \blacklozenge$ si $X \not\equiv t$ y $X \in var(t)$.
R9 $X /= X, S \,\square\, \sigma \rightsquigarrow_\chi \blacklozenge$

Cálculo de variables demandadas en $\mathcal{H}_{seq}$
Las siguientes reglas sirven para computar el conjunto $DVar_{\mathcal{H}_{seq}}(S)$ de variables demandadas por un conjunto satisfactible de restricciones primitivas $S \subseteq PCon(\mathcal{H}_{seq})$.

$DVar(seq\ t\ s \to! R, S) = \{R\} \cup (DVar_{\mathcal{H}_{seq}}(t == s, S\theta_1) \cap DVar(t /= s, S\theta_2))$
si R es una variable, $\theta_1 = \{R \mapsto true\}$ y $\theta_2 = \{R \mapsto false\}$.

$DVar(h\bar{t}_n == h\bar{s}_n, S) = DVar(t_1 == s_1, \ldots, t_n == s_n, S)$

$DVar(X == t, S) = \{X\} \cup var(t) \cup DVar(S\theta)$ donde $\theta = \{X \mapsto t\}$.

$DVar(h\bar{t}_n /= h'\bar{s}_m, S) = DVar(S)$, si $h \neq h'$ o $n \neq m$.

$DVar(h\bar{t}_n /= h\bar{s}_n, S) = DVar(t_1 /= s_1, S) \cap \ldots \cap DVar(t_n /= s_n, S)$

$DVar(X /= Y, S) = \{X, Y\} \cup DVar(S)$
$DVar(X /= t, S) = \{X\} \cup DVar(S)$, si t no es una variable.

Se asume que las últimas dos reglas no se pueden aplicar si una de las reglas previas es aplicable. En otro caso, el cómputo de $DVar(S)$ no funcionaría adecuadamente con estas reglas.

3.2.3 *CFLP($\mathcal{D}$)*-Programas

La clase de los *$CFLP(\mathcal{D})$-programas* son presentados como reglas de reescritura con restricciones que definen el comportamiento de funciones posiblemente de orden

superior y/o funciones no deterministas perezosas sobre el dominio $\mathcal{D}$, llamadas *reglas de programa*. Para ser más precisos, una regla de programa R para $CFLP(\mathcal{D})$ es de la forma $R : f\,\bar{t}_n \to r \Leftarrow P \,\square\, C$ y es necesario que satisfaga las condiciones que se listan a continuación:

1. El *lado izquierdo* $f\,\bar{t}_n$ es una expresión lineal, y para todo $1 \leq i \leq n$, $t_i \in Pat(\mathcal{U})$ son patrones totales.
2. El *lado derecho* $r \in Exp(\mathcal{U})$ es una expresión total.
3. P es una secuencia finita de *producciones* $e_i \to s_i$ $(1 \leq i \leq k)$ también destinada a ser interpretada como conjunción, y satisfaciendo las siguientes *condiciones de admisibilidad*:
   (a) Para todo $1 \leq i \leq k$, $e_i \in Exp(\mathcal{U})$ es una expresión total, $s_i \in Pat(\mathcal{U})$ es un patrón total y lineal, y $var(s_i) \cap var(f\,\bar{t}_n) = \emptyset$.
   (b) Es posible reordenar las producciones de P de la forma $P \equiv e_1 \to s_1, \ldots, e_k \to s_k$ donde $var(e_i) \cap var(s_j) = \emptyset$ para todo $1 \leq i \leq j \leq k$.
   (c) Para todo $1 \leq i < j \leq k$, $var(s_i) \cap var(s_j) = \emptyset$.
4. $C \subseteq DCon(\mathcal{D})$ es un conjunto finito de restricciones totales, destinadas a ser interpretadas como conjunción, y posiblemente incluyendo apariciones de símbolos definidos de función.

Una regla de programa tal que P y C son ambos vacíos puede ser abreviada como $f\,\bar{t}_n \to r$. Es posible observar que puede obtenerse una formulación equivalente para la condición de admisibilidad 3.(b) si definimos la *relación de produción* $X \gg_P Y$ si y sólo si hay algún $1 \leq i \leq k$ tal que $X \in var(e_i)$ y $Y \in var(s_i)$. Esta relación debe ser irreflexiva para el cierrre transitivo de $\gg_P$, o equivalentemente, un orden parcial estricto.

Ejemplo 39. *El siguiente $CFLP(\mathcal{D})$-programa puede ser usado en el dominio de restricciones $\mathcal{H}_{seq}$ presentado en el* Ejemplo 37. *Usamos las constructoras $0 \in DC^0$, $s \in DC^1$, una constructora para parejas (es decir, (e_1, e_2) denota la pareja de un primer elemento e_1 y un segundo elemento e_2) y una sintaxis para listas similar a la de Prolog ([ ] representa la lista vacía y $[X|Xs]$ representa la lista no vacía, que consta de un primer elemento X y un resto de lista Xs). Más ejemplos de $CFLP(\mathcal{D})$-programas pueden ser encontrados en [52].*

```
from   N        →  [N|from(s N)]
null   [ ]      →  s 0
null   [X|Xs]   →  0

split  []       →  ([], [])
split  [X|Xs]   →  case R X Ys Zs ⇐ split Xs → (Ys, Zs) □ seq X s 0 →! R

case   true   X   Ys   Zs   →  ([X|Ys], Zs)
case   false  X   Ys   Zs   →  (Ys, [X|Zs])
```

3.2.4 La Lógica de Reescritura con Restricciones $CRWL(\mathcal{D})$

La *Lógica de Reescritura con Restricciones* $CRWL(\mathcal{D})$, parametrizada por el dominio de restricciones $\mathcal{D}$, fue presentada en [52] a fin de proporcionar una semántica declarativa para los programas del esquema $CFLP(\mathcal{D})$. Con el fin de definir esta lógica, primero debemos introducir algunas nociones preliminares acerca de las sentencias que queremos extraer de un $CFLP(\mathcal{D})$-programa dado.

Definición 40 (Sentencias e Implicaciones $\mathcal{D}$). *Sea $\mathcal{D}$ cualquier dominio de restricciones fijado sobre un conjunto de elementos primitivos $\mathcal{U}$. Suponemos patrones parciales $t, t_i \in Pat_\perp(\mathcal{U})$, expresiones parciales $e, e_i \in Exp_\perp(\mathcal{U})$, y un conjunto finito $\Pi \subseteq PCon_\perp(\mathcal{D})$ de restricciones primitivas.*

1. *Consideramos dos posibles tipos de sentencias restringidas (*c-sentencias*):*
   - *(a)* c-producciones $e \to t \Leftarrow \Pi$. *Una c-producción se dice que es* trivial *si y sólo si $t = \perp$ o $InSat_{\mathcal{D}}(\Pi)$.*
   - *(b)* c-átomos $p\,\overline{e}_n \to\!! t \Leftarrow \Pi$, *con $p \in PF^n$ y t total. Un c-átomo se dice que es* trivial *si y sólo si $InSat_{\mathcal{D}}(\Pi)$.*
2. *Dadas dos c-sentencias φ y φ', decimos que φ $\mathcal{D}$-*implica *φ' (en símbolos, $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$) si y sólo si se cumple uno de los dos casos siguientes:*
   - *(a)* $\varphi = e \to t \Leftarrow \Pi$, $\varphi' = e' \to t' \Leftarrow \Pi'$, *y hay algún $\sigma \in Sub_\perp(\mathcal{U})$ tal que $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq e\sigma$, $\Pi' \models_{\mathcal{D}} t' \sqsubseteq t\sigma$.*
   - *(b)* $\varphi = p\,\overline{e}_n \to\!! t \Leftarrow \Pi$, $\varphi' = p\,\overline{e'}_n \to\!! t' \Leftarrow \Pi'$, *y hay algún $\sigma \in Sub_\perp(\mathcal{U})$ tal que $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} p\,\overline{e'}_n \sqsupseteq (p\,\overline{e}_n)\sigma$, $\Pi' \models_{\mathcal{D}} t' \sqsupseteq t\sigma$.* [1]

La siguiente definición supone un $CFLP(\mathcal{D})$-programa dado $\mathcal{P}$ y usa la notación $[\mathcal{P}]_\perp$ para el conjunto $\{R\theta \mid R \in \mathcal{P}, \theta \in Sub_\perp(\mathcal{U})\}$ formado por todas las posibles instancias de las reglas de definición de funciones pertenecientes a $\mathcal{P}$. El propósito del cálculo es inferir la validez semántica de c-sentencias semánticas para las reglas de programa en $\mathcal{P}$.

Definición 41 (Cálculo de Reescritura de Restricciones). *Escribimos $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ para indicar que la c-sentencia φ puede ser derivada de $\mathcal{P}$ en el cálculo de reescritura de restricciones $CRWL(\mathcal{D})$ usando las reglas de inferencia dadas en la* Figura 3.1. *Algunas de esas reglas dependen de las nociones semánticas dadas en la* Definición 35 *y de la siguiente noción semántica para producciones:*

$p\,\overline{t}_n \to t$ es una consecuencia *de Π en $\mathcal{D}$ (en símbolos, $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \to t$) si y sólo si $p^{\mathcal{D}}\,\overline{t}_n\eta \to t\eta$ se cumple para todo $\eta \in Sol_{\mathcal{D}}(\Pi)$.*

TI $\dfrac{}{\varphi}$ si φ es una c-sentencia trivial.

RR $\dfrac{}{t \to t \Leftarrow \Pi}$ si $t \in \mathcal{U} \cup \mathcal{V}$.

SP $\dfrac{}{s \to t \Leftarrow \Pi}$ si $s \in Pat_\bot(\mathcal{U})$, $s \in \mathcal{V}$ o $t \in \mathcal{V}$, y $\Pi \models_\mathcal{D} s \sqsupseteq t$.

DC $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_m \to t_m \Leftarrow \Pi}{h\overline{e}_m \to h\overline{t}_m \Leftarrow \Pi}$ si $h\overline{e}_m$ es pasiva.

IR $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_m \to t_m \Leftarrow \Pi}{h\overline{e}_m \to X \Leftarrow \Pi}$

si $h\overline{e}_m$ es pasiva pero no un patrón, $X \in \mathcal{V}$ y $\Pi \models_\mathcal{D} h\overline{t}_m \sqsupseteq X$.

PF $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_n \to t_n \Leftarrow \Pi}{p\,\overline{e}_n \to t \Leftarrow \Pi}$

si $p \in PF^n$, $t_i \in Pat_\bot(\mathcal{U})$ para todo $1 \le i \le n$, y $\Pi \models_\mathcal{D} p\,\overline{t}_n \to t$.

DF$_\mathcal{P}$ $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_n \to t_n \Leftarrow \Pi,\ P \,\square\, C \Leftarrow \Pi,\ r \to t \Leftarrow \Pi}{f\,\overline{e}_n \to t \Leftarrow \Pi}$

$$\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_n \to t_n \Leftarrow \Pi,\ P \,\square\, C \Leftarrow \Pi,\ r \to s \Leftarrow \Pi,\ s\,\overline{a}_k \to t \Leftarrow \Pi}{f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi}$$

si $f \in DF^n$ $(k > 0)$, $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, C) \in [\mathcal{P}]_\bot$, $s \in Pat_\bot(\mathcal{U})$.

AC $\dfrac{e_1 \to t_1 \Leftarrow \Pi, \ldots, e_n \to t_n \Leftarrow \Pi}{p\,\overline{e}_n \to\!! \; t \Leftarrow \Pi}$

si $p \in PF^n$, $t_i \in Pat_\bot(\mathcal{U})$ para todo $1 \le i \le n$, y $\Pi \models_\mathcal{D} p\,\overline{t}_n \to\!! \; t$.

Figura 3.1: Reglas de derivación para $CRWL(\mathcal{D})$.

[1] Obsérvese que $\Pi' \models_\mathcal{D} t' \sqsubseteq t\sigma$ sería incorrecto, porque $\to\!!$ se comporta monótonamente tanto en sus argumentos como en su resultado. Ver *Definición 3.(b)*.

Por convenio, acordaremos que ninguna regla de inferencia del calculo de reescritura con restricciones sea aplicada en el caso de que alguna regla anterior pueda aplicarse. Más concretamente, ninguna regla excepto **TI** puede ser usada para inferir una c-sentencia trivial, y **SP** no es aplicable siempre que **RR** sea aplicable. Además, también acordaremos que la premisa $P \,\square\, C \Leftarrow \Pi$ en la regla $\mathbf{DF}_{\mathcal{P}}$ debe ser entendida como una abreviatura para varias premisas $\alpha \Leftarrow \Pi$, una para cada sentencia atómica α que aparezca en $P \,\square\, C$.

Cualquier derivación en el calculo de reescritura de restricciones puede ser representada como un *árbol de prueba* cuyos nodos son etiquetados por c-sentencias, donde cada nodo ha sido inferido por sus hijos mediante las reglas de inferencia. De aquí en adelante, usaremos las siguientes notaciones:

1. T se dice que es un *árbol de prueba sencillo* si y sólo si en T no se usan las reglas de inferencia $\mathbf{DF}_{\mathcal{P}}$, **PF** ni **AC**.

2. $\mid T \mid$ sirve para denotar el *tamaño restringido* del árbol de prueba T, definido como el número de nodos en T que son inferidos con algunas de las reglas $\mathbf{DF}_{\mathcal{P}}$, **PF** o **AC**. Obviamente, $\mid T \mid = 0$ si y sólo si T es un árbol de prueba sencillo.

3. $T : \mathcal{P} \vdash_{\mathcal{D}} \varphi$ indica que $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ sirve como testigo del árbol de prueba T.

El siguiente resultado establece dos propiedades útiles para el cálculo de reescritura de restricciones. La demostración y otras propiedades de $CRWL(\mathcal{D})$ pueden encontrarse en [52].

Lema 42 (Propiedades del Cálculo $CRWL(\mathcal{D})$).

1. ***Propiedad de Aproximación:*** *Para cualquier $e \in Exp_{\perp}(\mathcal{U})$, $t \in Pat_{\perp}(\mathcal{U})$: $\Pi \models_{\mathcal{D}} e \sqsupseteq t$ si y sólo si hay algún árbol de prueba sencillo T tal que $T : \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$ (derivación a partir de un programa vacío).*

2. ***Propiedad de Implicación:*** *$T : \mathcal{P} \vdash_{\mathcal{D}} \varphi$ y $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$ implica $T' : \mathcal{P} \vdash_{\mathcal{D}} \varphi'$ para algún árbol de prueba T' tal que $\mid T' \mid \leq \mid T \mid$.*

Los resultados de corrección de la deducibilidad en $CRWL(\mathcal{D})$ con respecto a una semántica adecuada de modelos se presentan en [52]. De forma más precisa, como se demuestra en [52], $CRWL(\mathcal{D})$ es correcto y completo con respecto a *semánticas fuertes*, y correcto y completamente cerrado con respecto a *semánticas débiles*, dos diferentes clases de semánticas.

Ejemplo 43. *Dado el siguiente programa escrito en $CFLP(\mathcal{R})$:*

$inInterval :: real \to real \to real \to boolean$
$inInterval\ A\ B\ X\ \to\ and\ (A \leq X)\ (X \leq B)$

$from :: real \to real \to [real]$
$from\ X\ D\ \to\ [X \mid from\ (X + D)\ D]$

$takeWhile :: (real \to bool) \to [real] \to [real]$
$takeWhile\ P\ [\,]\ \to\ [\,]$
$takeWhile\ P\ [X \mid Xs]\ \to\ if\ (P\ X)\ [X \mid takeWhile\ P\ Xs]$

$if\ true\ L\ R\ \to\ L$
$if\ false\ L\ R\ \to\ R$

Describen las funciónes "inInterval" que dado tres reales, indica si se cumple que el tercero es mayor o igual que el primero y menor o igual que el segundo, la función "from" genera una lista infinita comenzando desde un elemento dado por el primer argumento X, y el siguiente elemento de la lista tiene un incremento D con respecto al anterior, la función "takeWhile" que va tomando todos los elementos de una lista hasta que encuentra un elemento que no cumple la propiedad P, y la función auxiliar "if", que dado un booleano, si es 'true' devuelve el segundo argumento, y si es 'false', el tercero.

Tenemos que el árbol de prueba T correspondiente a la CRWL($\mathcal{D}$)-derivación

$T : \mathcal{P} \vdash_{\mathcal{R}} takeWhile\ (inInterval\ 0\ 1)\ (from\ X\ 0.5) == [X] \Leftarrow X > 0.5,\ X \leq 1$

tiene la siguiente estructura (hemos separado la definición del subárbol de T, T_0 para una mayor sencillez en la lectura):

AC $takeWhile\,(\,inInterval\ 0\ 1)(from\ X\ 0.5) == [X] \Leftarrow X > 0.5\,,\ X \leq 1$
  $X > 0.5\,,\ X \leq 1 \models_{\mathbb{R}} [X] == [X]$
  DC $[X] \to [X] \Leftarrow X > 0.5\,,\ X \leq 1$
    RR $X \to X \Leftarrow X > 0.5\,,\ X \leq 1$
    DC $[\,] \to [\,] \Leftarrow X > 0.5\,,\ X \leq 1$
  DF $takeWhile\,(\,inInterval\ 0\ 1)(from\ X\ 0.5) \to [X] \Leftarrow X > 0.5\,,\ X \leq 1 \leq 1$
    DC $inInterval\ 0\ 1 \to interval01 \Leftarrow X > 0.5\,,\ X \leq 1$
      RR $0 \to 0 \Leftarrow X > 0.5\,,\ X \leq 1$
      RR $1 \to 1 \Leftarrow X > 0.5\,,\ X \leq 1$
    DF $from\ X\ 0.5 \to [X, X + 0.5|\bot] \Leftarrow X > 0.5\,,\ X \leq 1$
      RR $X \to X \Leftarrow X > 0.5\,,\ X \leq 1$
      RR $0.5 \to 0.5 \Leftarrow X > 0.5\,,\ X \leq 1$
      DC $[X|from\ (X\ +\ 0.5)\ 0.5] \to [X, X + 0.5|\bot]$

$\Leftarrow X > 0.5\,,\ X \leq 1$
RR $X \to X \Leftarrow X > 0.5\,,\ X \leq 1$
DF $from\ (X + 0.5)\ 0.5 \to [X + 0.5|\bot]$
$\Leftarrow X > 0.5\,,\ X \leq 1$
PF $X + 0.5 \to X + 0.5 \Leftarrow X > 0.5\,,\ X \leq 1$
RR $X \to X \Leftarrow X > 0.5\,,\ X \leq 1$
RR $0.5 \to 0.5 \Leftarrow X > 0.5\,,\ X \leq 1$
$X > 0.5\,,\ X \leq 1 \models_{\mathbb{R}} X + 0.5 \to X + 0.5$
RR $0.5 \to 0.5 \Leftarrow X > 0.5\,,\ X \leq 1$
DC $[X + 0.5|from\ ((X + 0.5) + 0.5)\ 0.5]$
$\to [X + 0.5|\bot] \Leftarrow X > 0.5\,,\ X \leq 1$
PF $X + 0.5 \to X + 0.5 \Leftarrow X > 0.5\,,\ X \leq 1$
TI $from\ ((X + 0.5) + 0.5)\ 0.5 \to \bot$
$\Leftarrow X > 0.5\,,\ X \leq 1$
DF $if\ (inInterval\ 0\ 1\ X)\ [X|takeWhile\ (inInterval\ 0\ 1)\ [X + 0.5|\bot]]\ [\,]$
$\to [X] \Leftarrow X > 0.5\,,\ X \leq 1$
DF $inInterval\ 0\ 1\ X \to true \Leftarrow X > 0.5\,,\ X \leq 1$
RR $0 \to 0 \Leftarrow X > 0.5\,,\ X \leq 1$
RR $1 \to 1 \Leftarrow X > 0.5\,,\ X \leq 1$
RR $X \to X \Leftarrow X > 0.5\,,\ X \leq 1$
DF $and\ (0 \leq X)\ (X \leq 1) \to true \Leftarrow X > 0.5\,,\ X \leq 1$
PF $0 \leq X \to true \Leftarrow X > 0.5\,,\ X \leq 1$
RR $0 \to 0 \Leftarrow X > 0.5\,,\ X \leq 1$
RR $X \to X \Leftarrow X > 0.5\,,\ X \leq 1$
$X > 0.5, X \leq 1 \models_{\mathbb{R}} 0 \leq X \to true$
PF $X \leq 1 \to true \Leftarrow X > 0.5\,,\ X \leq 1$
RR $X \to X \Leftarrow X > 0.5\,,\ X \leq 1$
RR $1 \to 1 \Leftarrow X > 0.5\,,\ X \leq 1$
$X > 0.5, X \leq 1 \models_{\mathbb{R}} X \leq 1 \to true$
DC $true \to true \Leftarrow X > 0.5\,,\ X \leq 1$
DC $[X|takeWhile\ (inInterval\ 0\ 1)\ [X + 0.5|\bot]] \to [X]$
$\Leftarrow X > 0.5\,,\ X \leq 1$
RR $X \to X \Leftarrow X > 0.5\,,\ X \leq 1$
DC $takeWhile\ (inInterval\ 0\ 1)\ [X+0.5|\bot] \to [\,] \Leftarrow X > 0.5\,,\ X \leq 1$
$\ldots T_0 \ldots$
DC $[\,] \to [\,] \Leftarrow X > 0.5\,,\ X \leq 1$
DC $[X] \to [X] \Leftarrow X > 0.5\,,\ X \leq 1$
RR $X \to X \Leftarrow X > 0.5\,,\ X \leq 1$
DC $[\,] \to [\,] \Leftarrow X > 0.5\,,\ X \leq 1$

El árbol de prueba T_0 *es tal que* $T_0 : \mathcal{P} \vdash_{\mathbb{R}} takeWhile\ (inInterval\ 0\ 1)[X+0.5|\bot] \to [\,] \Leftarrow X > 0.5\,,\ X \leq 1$

Adicionalmente, el siguiente resultado será útil para demostrar propiedades sobre el cálculo de estrechamiento $CDNC(\mathcal{D})$ y el nuevo cálculo concurrente en el Capítulo 4. Su demostración puede encontrarse en el Apéndice A.

Lema 44 (Propiedad de Particionado). *Sea $\mathcal{P}$ un $CFLP(\mathcal{D})$-programa. Para cualquier $e \in Exp_\perp(\mathcal{U})$, $t \in Pat_\perp(\mathcal{U})$ y $p \in Pos(e)$, las siguientes condiciones son equivalentes:*

1. $T : \mathcal{P} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$.
2. *Existe $s \in Pat_\perp(\mathcal{U})$ tal que $T_1 : \mathcal{P} \vdash_{\mathcal{D}} e|_p \to s \Leftarrow \Pi$ y $T_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p \to t \Leftarrow \Pi$.*

Además, cuando probamos "(1) $\Rightarrow$ (2)" uno puede escoger $\mid T_1 \mid, \mid T_2 \mid \leq \mid T \mid$.

3.3 La clase de los $COISS(\mathcal{D})$-Programas con Árboles Definicionales

La clase de los llamados $COISS(\mathcal{D})$-programas con restricciones y árboles definicionales usada en esta sección es una subclase de los $CFLP(\mathcal{D})$-programas genéricos presentados en [50, 52]. En esta sección introduciremos de los $COISS(\mathcal{D})$-programas y sus semánticas pretendidas.

3.3.1 Árboles Definicionales con Solapamiento y Restricciones

Los *$CFLP(\mathcal{D})$-programas* son presentados como reglas de reescritura con restricciones que definen el comportamiento de funciones de orden superior y/o funciones perezosas no deterministas sobre $\mathcal{D}$ llamadas reglas de programa. Más concretamente, una regla de programa R para $f \in DF^n$ es de la forma $R : f\,\bar{t}_n \to r \Leftarrow P \,\square\, C$ (abreviado como $f\,\bar{t}_n \to r$ si P y C son ambos vacíos) y es necesario que se cumplan las tres condiciones que se listan a continuación:

1. El *lado izquierdo de la regla* $f\,\bar{t}_n$ es una expresión lineal, y para todo $1 \leq i \leq n$, $t_i \in Pat(\mathcal{U})$ son patrones totales. El *lado derecho* $r \in Exp(\mathcal{U})$ también es total.
2. P es una secuencia infinita de *producciones* de la forma $e_i \to R_i$ $(1 \leq i \leq k)$, que se interpreta como conjunción de definiciones locales, y cumpliendo las siguientes *condiciones de admisibilidad*:
   (a) Para todo $1 \leq i \leq k$, $e_i \in Exp(\mathcal{U})$ es una expresión total, R_i es una variable diferente, y $R_i \notin var(f\,\bar{t}_n)$.
   (b) Es posible reordenar las producciones de P de la siguiente manera $P \equiv e_1 \to R_1, \ldots, e_k \to R_k$ donde $R_j \notin var(e_i)$ para todo $1 \leq i \leq j \leq k$ (*sin ciclos*).

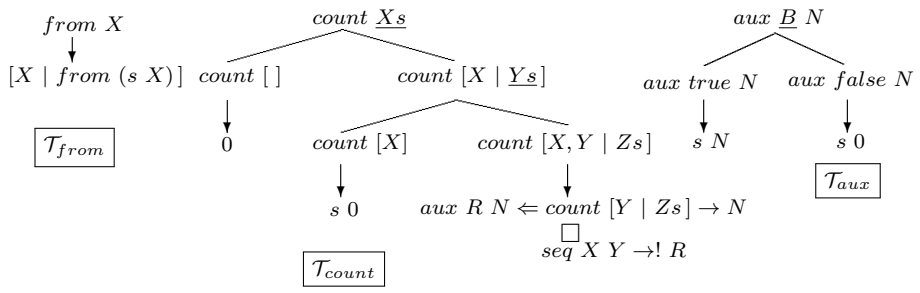

Figura 3.2: Árboles definicionales con restricciones para $from$, $count$ y aux.

3. C es un conjunto finito de restricciones totales, también interpretadas como conjunción, y posiblemente incluyendo apariciones de símbolos de funciones definidas.

Ejemplo 45. *El siguiente $CFLP(\mathcal{D})$-programa puede usarse sobre el dominio de restricciones $\mathcal{H}_{seq}$. Usamos las construtoras $0 \in DC^0$, $s \in DC^1$, y la sintaxis de listas similar a la de Prolog.*

$$
\begin{array}{llllll}
from & X & \to & [X \mid from\ (s\ X)] & & \%\ \textit{crea una lista infinita a partir de un valor inicial } X. \\
count & [\,] & \to & 0 & & \%\ \textit{Cuenta los valores repetidos consecutivos desde} \\
count & [X] & \to & s\ 0 & & \%\ \textit{la cabeza de la lista hasta que encuentre un valor} \\
count & [X,Y|Zs] & \to & aux\ R\ N & \Leftarrow\ count\ [Y|Zs] \to N\ \square\ seq\ X\ Y \to!\ R & \%\ \textit{diferente.} \\
aux & true & N & \to & s\ N & \\
aux & false & N & \to & s\ 0 &
\end{array}
$$

La clase de los $COISS(\mathcal{D})$-programas queda definida como una subclase de los $CFLP(\mathcal{D})$-programas donde las reglas definidas pueden organizarse en una estructura jerárquica llamada *árbol definicional* [4]. Más concretamente, hemos escogido reformular las nociones presentadas en [5, 6, 21] sobre *árboles definicionales con solapamiento* y *sistemas inductivamente secuenciales con solapamiento*, incluyendo reglas de restricciones sobre un dominio genérico de restricciones $\mathcal{D}$.

Definición 46 (Árboles definicionales con restricciones). *Sea $\mathcal{P}$ un $CFLP(\mathcal{D})$-programa sobre un dominio de restricciones $\mathcal{D}$ dado. Un* patrón de llamada *es cualquier patrón lineal de la forma $f\bar{t}_n$, donde $f \in DF^n$ y $\bar{t}_n \in Pat_\perp(\mathcal{U})$. $\mathcal{T}$ es un* Árbol Definicional con Restricciones sobre $\mathcal{D}$ *(cDT($\mathcal{D}$) para abreviar)* con patrón de llamada τ *si y solo si su altura es finita y se cumple alguno de los dos casos siguientes:*

- $\mathcal{T} \equiv \underline{rule}(\tau \to r_1 \Leftarrow P_1 \,\square\, C_1 | \ldots | r_m \Leftarrow P_m \,\square\, C_m)$, *donde $\tau \to r_i \Leftarrow P_i \,\square\, C_i$ para todo $1 \leq i \leq m$ es una variante de una regla de programa en $\mathcal{P}$.*

- $\mathcal{T} \equiv \underline{case}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$, *donde* X *es una variable en* τ, $h_1, \ldots, h_k$ $(k > 0)$ *son símbolos pasivos de* $\mathcal{P}$ *diferentes dos a dos, y para todo* $1 \leq i \leq k$, $\mathcal{T}_i$ *es un* $cDT(\mathcal{D})$ *con patrón de llamada* $\tau\sigma_i$, *donde* $\sigma_i = \{X \mapsto h_i\overline{Y}_{m_i}\}$ *con* $\overline{Y}_{m_i}$ *variables nuevas y distintas tal que* $h_i\overline{Y}_{m_i} \in Pat(\mathcal{U})$.

Representamos un $cDT(\mathcal{D})$ $\mathcal{T}$ *con patrón de llamada* τ *usando la notación* $\mathcal{T}_\tau$. *Un* $cDT(\mathcal{D})$ de un símbolo de función $f \in DF^n$ *definido por* $\mathcal{P}$ *es un* $cDT(\mathcal{D})$ $\mathcal{T}$ *con patrón de llamada* $f\overline{X}_n$, *donde* $\overline{X}_n$ *son variables nuevas. Lo representamos usando la notación* $\mathcal{T}_f$.

Definición 47 ($COISS(\mathcal{D})$**-programas).**

1. *Un símbolo de función* $f \in DF^n$ *se dice que es* inductivamente secuencial con solapamiento y restricciones *con respecto a un* $CFLP(\mathcal{D})$*-programa* $\mathcal{P}$ *si y solo si existe un* $cDT(\mathcal{D})$ $\mathcal{T}_f$ *of* f *tal que la colección de todas las reglas del programa* $\tau \to r_i \Leftarrow P_i \square C_i$ $(1 \leq i \leq m)$ *obtenidas de los diferentes nodos* $\underline{rule}(\tau \to r_1 \Leftarrow P_1 \square C_1 | \ldots | r_m \Leftarrow P_m \square C_m)$ *que aparecen en* $\mathcal{T}_f$ *son iguales, salvo variantes, a la colección de todas las reglas del programa en* $\mathcal{P}$ *cuyo lado izquierdo tiene el símbolo raíz* f.

2. *Un* $CFLP(\mathcal{D})$*-programa* $\mathcal{P}$ *se dice que es un* sistema inductivamente secuencial con solapamiento y restricciones sobre $\mathcal{D}$ *(*$COISS(\mathcal{D})$*, del inglés* Constrained Overlapping Inductively Sequential System*) si y solo si cada función definida en* $\mathcal{P}$ *es inductivamente secuencial con solapamiento y restricciones.*

Como ejemplo concreto, podemos considerar el $CFLP(\mathcal{H}_{seq})$-programa dado en el *Ejemplo 45*. A partir de los árboles definicionales ilustrados en la *Figura 3.2*, es fácil probar que este programa es un $COISS(\mathcal{H}_{seq})$. Por ejemplo, el símbolo de función definida *count* tiene el siguiente árbol definicional $\mathcal{T}_{count}$:

$$
\begin{array}{l}
\underline{case}\ (count\ Xs, Xs, [\\
\quad \underline{rule}\ (count\ [\,]\ \to\ 0),\\
\quad \underline{case}\ (count\ [X|Ys], Ys, [\\
\qquad \underline{rule}\ (count\ [X] \to s\ 0),\\
\qquad \underline{rule}\ (count\ [X, Y|Zs]\ \to\ aux\ R\ N\ \Leftarrow count\ [Y|Zs]\ \to\ N\ \square\ seq\ X\ Y\ \to!\ R)\ ]\ )\ ]\ )
\end{array}
$$

3.4 Estrechamiento Perezoso con Restricciones y Árboles Definicionales

En este apartado presentamos un *Cálculo de Restricciones de Estrechamiento Demandado* $CDNC(\mathcal{D})$ (del inglés *Constrained Demanded Narrowing Calculus*), sobre $COISS(\mathcal{D})$-programas. Para nuestra discusión sobre este nuevo cálculo con árboles definicionales vamos a combinar las ideas y técnicas principales del $CLNC(\mathcal{D})$

cálculo en [50] (con restricciones genéricas pero sin árboles definicionales) y el DNC-cálculo en [21] (con árboles definicionales, pero sin restricciones genéricas). La idea general es asegurar la computación de respuestas para los objetivos que sean correctas con respecto a la semántica $CRWL(\mathcal{D})$, mientras que se usan los árboles definicionales de forma similar a [7] para asegurar que todas los pasos de estrechamiento perezosos con restricciones que ocurren durante la computación son necesarios. Primero daremos una definición precisa para la clase de objetivos y respuestas admisibles que vamos a usar.

3.4.1 Objetivos y Respuestas

Un *objetivo* para un $COISS(\mathcal{D})$-programa debe ser de la forma $G \equiv \exists \overline{U}.P \,\square\, C \,\square\, S \,\square\, \sigma$, donde el símbolo '$\square$' es interpretado como conjunción, y:

- $\overline{U} =_{def} evar(G)$ es el conjunto de *variables existenciales* de el objetivo G. Son variables intermedias, cuyas vinculaciones serían patrones parciales.
- $P \equiv e_1 \to R_1, \ldots, e_n \to R_n$ es una conjunción finita de producciones donde cada R_i es una variable distinta, y e_i es una expresión o un par de la forma $< \tau, \mathcal{T} >$, donde τ es una instancia del patrón en la raíz de un $cDT(\mathcal{D})$ $\mathcal{T}$. Esas producciones $e \to R$ cuyo lado izquierdo e es solamente una expresión son denominadas *suspensiones*, mientras que las que cuyo lado izquierdo es de la forma $< \tau, \mathcal{T} >$ son denominadas *producciones demandadas*. El conjunto de *variables producidas* de G se define como $pvar(P) =_{def} \{R_1, \ldots, R_n\}$ y definimos la *relación de producción* $X \gg_P Y$ si y solo si hay algún $1 \leq i \leq n$ tal que $X \in var(e_i)$ e $Y \equiv R_i$.
- $C \equiv \delta_1, \ldots, \delta_k$ es una conjunción finita de restricciones totales (posiblemente con apariciones de símbolos de funciones definidas).
- $S \equiv \pi_1, \ldots, \pi_l$ es una conjunción finita de restricciones primitivas totales, llamada *almacén de restricciones*.
- σ es una sustitución idempotente llamada *sustitución respuesta* tal que $Dom(\sigma) \cap var(P \,\square\, C \,\square\, S) = \emptyset$.

Además, cualquier objetivo admisible debe ser satisfecho por las mismas condiciones de admisibilidad dadas en [50] sobre variables producidas ademas de una nueva condición de admisibilidad para árboles definicionales:

DT *Para cada producción demandada $< \tau, \mathcal{T} > \to R$ en P, la variable R es demandada (es decir $R \in DVar_{\mathcal{D}}(G)$ en el sentido de la* Definición 48*), y las variables en $\mathcal{T}$ no aparecen en ningún otro lugar del objetivo.*

De forma similar a [21, 50], $CDNC(\mathcal{D})$ usa una noción de *variable demandada* para tratar con la evaluación perezosa, pero ahora en este trabajo con respecto a un almacén de restricciones, orden superior y árboles definicionales con restricciones.

Definición 48 (**Variables Demandadas**). *Sea* $G \equiv \exists\overline{U}.\ P \square C \square S \square \sigma$ *un objetivo admisible para un* $COISS(\mathcal{D})$*-programa dado y* $X \in var(G)$*. Decimos que* X *es una* variable demandada *en* G *si y solo si se cumple uno de los siguientes casos:*

- $X \in DVar_{\mathcal{D}}(S)$ *(ver item* 1 *en la* Definición 36*).*
- *Existe alguna suspensión* $(X\overline{a}_k \to R) \in P$ *tal que* $k > 0$ *y* R *es una variable demandada en* G*.*
- *Existe alguna producción demandada* $(< e,\ \underline{case}\ (\tau, Y, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) > \to R) \in P$ *tal que* $X = e|_{pos(Y,\tau)}$ *y* R *es una variable demandada en* G*.*

Escribimos $DVar_{\mathcal{D}}(G)$ *(o de forma más precisa,* $DVar_{\mathcal{D}}(P \square S)$*) para el conjunto de variables demandadas en el objetivo* G*.*

Un objetivo admisible $G \equiv \exists\overline{U}.\ P \square C \square S \square \sigma$ se dice que es un *objetivo resuelto* si y solo si P y C son vacíos y S está en $\emptyset$-forma resuelta en el sentido del item 2.(a) de la *Definición 36*. Un *objetivo inicial* puede ser cualquier objetivo admisible.

Definición 49 (**Respuestas**). *Una* respuesta *para un objetivo admisible* $G \equiv \exists\overline{U}.\ P \square C \square S \square \sigma$ *y un* $COISS(\mathcal{D})$*-programa dado* $\mathcal{P}$*, debe ser de la forma* $\Pi \square \theta$*, donde* $\Pi \subseteq PCon(\mathcal{D})$ *es una conjunción finita de restricciones primitivas totales,* $\theta \in Sub_{\perp}(\mathcal{U})$ *es una sustitución idempotente tal que* $Dom(\theta) \cap var(\Pi) = \emptyset$*, y hay alguna sustitución* $\theta' =_{\backslash evar(G)} \theta$ *satistaciendo las tres condiciones siguientes:*

- $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\theta' \Leftarrow \Pi$ *en* $CRWL(\mathcal{D})$ *(para producciones demandadas* $(< \tau, \mathcal{T} > \to R) \in P$*, consideraremos solo* $\mathcal{P} \vdash_{\mathcal{D}} \tau\theta' \to \theta'(R) \Leftarrow \Pi$*, porque los árboles definicionales se usan únicamente para controlar eficientemente los cómputos),*
- $\Pi \models_{\mathcal{D}} S\theta'$ *(es decir,* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{D}}(S\theta')$ *de acuerdo con el tercer item de la* Definición 34*),*
- $X\theta' \equiv t\theta'$ *para cada vínculo* $X \mapsto t \in \sigma$*, que abreviamos como* $\theta' \in Sol(\sigma)$*.*

Denotamos $Ans_{\mathcal{P}}(G)$ *al conjunto de todas las posibles respuestas de* G*. Una respuesta* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$ *se dice que es* trivial *si y solo si* $InSat_{\mathcal{D}}(\Pi)$ *y* no trivial *en cualquier otro caso. Por otra parte, podemos relacionar objetivos y respuestas extendiendo la noción de solución presentada en la* Definición 34*.*

Definición 50 (Soluciones). *Sea* $G \equiv \exists\overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma$ *un objetivo admisible para un* $COISS(\mathcal{D})$*-programa* $\mathcal{P}$ *dado. Diremos que una valoración* $\mu \in Val_{\perp}(\mathcal{D})$ *es una* solución *de* G *si y solo si* $(\emptyset \,\square\, \mu) \in Ans_{\mathcal{P}}(G)$*. Escribimos* $Sol_{\mathcal{P}}(G)$ *para representar el conjunto de todas las soluciones de* G*. Análogamente, definimos el conjunto de soluciones para una respuesta* $\Pi \,\square\, \theta$ *como* $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) =_{def} \{\mu \in Val_{\perp}(\mathcal{D}) \mid \mu \in Sol_{\mathcal{D}}(\Pi) \cap Sol(\theta)\}$*.*

El siguiente resultado, cuya demostración se encuentra en el Apéndice A, resulta útil para probar las principales propiedades del cálculo $CDNC(\mathcal{D})$ y muestra que la semántica $CRWL(\mathcal{D})$ no acepta valores indefinidos para variables demandadas, identificando los pasos de cómputo demandados y necesarios en las derivaciones.

Lema 51 (Lema de Demanda). *Si* $\Pi \,\square\, \theta$ *es una respuesta no trivial para un objetivo admisible* G *de un* $COISS(\mathcal{D})$*-programa y* $X \in DVar_{\mathcal{D}}(G)$ *entonces* $\theta'(X) \neq \perp$ *para toda* $\theta' =_{\setminus evar(G)} \theta$ *dada en la* Definición 49*.*

3.4.2 El Cálculo $CDNC(\mathcal{D})$

El cálculo $CDNC(\mathcal{D})$ está formado por un conjunto de reglas de transformación para objetivos admisibles. Cada transformación es de la forma $G \Vdash G'$, especificando una de las posibles maneras de realizar un paso de resolución objetivos. Escribimos $G \vdash_R G'$ para indicar que $G \Vdash G'$ por medio de la $CDNC(\mathcal{D})$-regla de transformación R. Las derivaciones son secuencias de $\Vdash$-pasos. Como en el caso de las SLD-derivaciones con restricciones para $CLP(\mathcal{D})$-programas [36], las derivaciones con éxito terminarán eventualmente con un objetivo resuelto. Derivaciones con fallo (terminadas con un objetivo inconsistente $\blacksquare$) y derivaciones infinitas también son posibles. De forma similar a [21, 50], todas las reglas de transformación de objetivos son aplicadas sobre P y C como conjuntos, más que como secuencias.

- Las reglas de transformaciones concernientes a suspensiones $e \to R$ (ver *Figura 3.3*) están diseñadas con el propósito de modelar el comportamiento del estrechamiento con restricciones perezoso con *sharing* como en el $CLNC(\mathcal{D})$-cálculo [50], suponiendo funciones primitivas, posiblemente funciones definidas de orden superior y variables funcionales.

- Las reglas de transformación de objetivos para producciones demandadas $< e, \mathcal{T} > \to R$ (ver *Figura 3.4*) codifican la *estrategia de estrechamiento demandado* guiada por el árbol $\mathcal{T}$, de forma similar a [21, 50]: si $\mathcal{T}$ es un $\underline{rule}$-árbol, entonces la transformación **RRA** escoge una de las reglas disponibles para la reescritura de e, introduciendo suspensiones y restricciones apropiadas en el nuevo objetivo tal que la evaluación perezosa esté asegurada. Si $\mathcal{T}$ es un $\underline{case}$-árbol, una de las transformaciones **CSS**, **DI** o **DN** puede ser aplicada, de acuerdo al tipo de símbolo que aparece en e en la posición de distinción de

SS Suspensión Simple $\exists X, \overline{U}.\ t \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{SS}}$
$\exists \overline{U}.\ (P \square C \square S)\sigma_0 \square \sigma$ si $t \in Pat(\mathcal{U})$ y $\sigma_0 = \{X \mapsto t\}$.

IM Imitación $\exists X, \overline{U}.\ h\overline{e}_m \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{IM}}$
$\exists \overline{X}_m, \overline{U}.\ (\overline{e_m \to X_m}, P \square C \square S)\sigma_0 \square \sigma$
si $h\overline{e}_m \notin Pat(\mathcal{U})$ es pasiva, $X \in DVar_{\mathcal{D}}(P \square S)$ y $\sigma_0 = \{X \mapsto h\overline{X}_m\}$ con $\overline{X}_m$ variables nuevas tales que $h\overline{X}_m \in Pat(\mathcal{U})$.

EL Eliminación $\exists X, \overline{U}.\ e \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{EL}}$
$\exists \overline{U}.\ P \square C \square S \square \sigma$ si $X \notin var(P \square C \square S \square \sigma)$.

PF Función Primitiva $\exists X, \overline{U}.\ p\overline{e}_n \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{PF}}$
$\exists \overline{X}_q, X, \overline{U}.\ \overline{e_q \to X_q}, P \square C \square p\overline{t}_n \to! X, S \square \sigma$
si $p \in PF^n$, $X \in DVar_{\mathcal{D}}(P \square S)$, y $\overline{X}_q$ son nuevas variables ($0 \leq q \leq n$ es el número de $e_i \notin Pat(\mathcal{U})$) tales que $t_i \equiv X_j$ $(0 \leq j \leq q)$ si $e_i \notin Pat_{\perp}(\mathcal{U})$ y $t_i \equiv e_i$ en cualquier otro caso para todo $1 \leq i \leq n$.

DT Árbol Definicional $\exists X, \overline{U}.\ f\overline{e}_n \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{DT_1}}$
$\exists X, \overline{U}.\ < f\overline{e}_n, \mathcal{T}_{f\overline{X}_n} > \to X, P \square C \square S \square \sigma$
$\exists X, \overline{U}.\ f\overline{e}_n\overline{a}_k \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{DT_2}}$
$\exists X, X', \overline{U}.\ < f\overline{e}_n, \mathcal{T}_{f\overline{X}_n} > \to X', X'\overline{a}_k \to X, P \square C \square S \square \sigma$
si $f \in DF^n (k > 0)$, $X \in DVar_{\mathcal{D}}(P \square S)$, ambos X' y todas las variables en $\mathcal{T}_{f\overline{X}_n}$ son nuevas variables.

FV Variable Funcional $\exists X, \overline{U}.\ F\overline{e}_q \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{FV}}$
$\exists \overline{X}_p, X, \overline{U}.\ (h\overline{X}_p\overline{e}_q \to X, P \square C \square S)\sigma_0 \square \sigma\sigma_0$
si $F \notin pvar(P), q > 0$, $X \in DVar_{\mathcal{D}}(P \square S)$, $\sigma_0 = \{F \mapsto h\overline{X}_p\}$ y $\overline{X}_p$ son nuevas variables tales que $h\overline{X}_p \in Pat(\mathcal{U})$.

Figura 3.3: $CDNC(\mathcal{D})$-reglas para suspensiones.

casos. Por lo demás, fallamos usando **CC**. En caso contrario, la computación debe retrasarse.

- Las reglas de transformación de objetivos concernientes a las restricciones (ver *Figura 3.5*) están diseñadas para combinar restricciones (primitivas o definidas por el usuario) con la acción de un resolutor de restricciones que satisface los requerimientos dados en la *Definición 36*. La regla de fallo **SF** se usa para detectar errores en la resolución de restricciones.

El siguiente ejemplo de resolución de un objetivo pretende ilustrar las principales propiedades del cálculo $CDNC(\mathcal{D})$. Para cada paso de transformación de objetivos, subrayaremos qué subobjetivo ha sido seleccionado.

CSS Selección de Caso

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \square C \square S \square \sigma \vdash_{\mathbf{CSS}} \exists R, \overline{U}.\ < e, \mathcal{T}_i >\!\!\rightarrow R, P \square C \square S \square \sigma$$

Si $e|_{pos(X,\tau)} = h_i \ldots$, con $1 \leq i \leq k$ dado por e, donde h_i es el símbolo pasivo asociado a $\mathcal{T}_i$.

DI Instanciación Demandada

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \square C \square S \square \sigma \vdash_{\mathbf{DI}} \exists \overline{Y}_{m_i}, R, \overline{U}.\ (< e, \mathcal{T}_i >\!\!\rightarrow R, P \square C \square S)\sigma_0 \square \sigma\sigma_0$$

Si $e|_{pos(X,\tau)} = Y$, $Y \notin pvar(P)$, $\sigma_0 = \{Y \mapsto h_i \overline{Y}_{m_i}\}$ con h_i $(1 \leq i \leq k)$ es el símbolo pasivo asociado a $\mathcal{T}_i$ e $\overline{Y}_{m_i}$ son nuevas variables.

DN Estrechamiento Demandado

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \square C \square S \square \sigma \vdash_{\mathbf{DN}} \exists R', R, \overline{U}.\ e|_{pos(X,\tau)} \rightarrow R', < e[R']_{pos(X,\tau)}, \underline{case}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \square C \square S \square \sigma$$

Si $e|_{pos(X,\tau)} = g \ldots$ con $g \in FS$ activo (símbolo primitivo o de función definida) y R' una variable nueva.

RRA Aplicación de Regla de Reescritura

$$\exists R, \overline{U}.\ < e, \underline{rule}(\tau \rightarrow r_1 \Leftarrow P_1 \square C_1 | \ldots | r_k \Leftarrow P_k \square C_k) >\!\!\rightarrow R, P \square\ C \square S \square \sigma \vdash_{\mathbf{RRA}} \exists \overline{X}, R, \overline{U}.\ \sigma_f(R_1) \rightarrow R_1, \ldots, \sigma_f(R_m) \rightarrow R_m, r_i\sigma_c \rightarrow R, P_i\sigma_c, P \square C_i\sigma_c, C \square S \square \sigma$$

- $\sigma_0 = \sigma_c \uplus \sigma_f$ con $Dom(\sigma_0) = var(\tau)$ y $\tau\sigma_0 = e$.
- $\sigma_c =_{def} \sigma \restriction_{dom_c(\sigma_0)}$, donde $dom_c(\sigma_0) = \{X \in Dom(\sigma_0) \mid \sigma_0(X) \in Pat(\mathcal{U})\}$.
- $\sigma_f =_{def} \sigma \restriction_{dom_f(\sigma_0)}$, donde $dom_f(\sigma_0) = \{X \in Dom(\sigma_0) \mid \sigma_0(X) \notin Pat(\mathcal{U})\} = \{R_1, \ldots, R_m\}$.
- $\overline{X} \equiv var(\tau \rightarrow r_i \Leftarrow P_i \square C_i) \backslash dom_c(\sigma_0)$.

CC Caso no Cubierto

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \square C \square S \square \sigma \vdash_{\mathbf{CC}} \blacksquare$$

Si $e|_{pos(X,\tau)} = h \ldots$ es un símbolo pasivo y $h \notin \{h_1, \ldots, h_k\}$, donde h_i es el símbolo pasivo asociado a $\mathcal{T}_i$ $(1 \leq i \leq k)$.

Figura 3.4: $CDNC(\mathcal{D})$-reglas para producciones demandadas y detección de errores.

Ejemplo 52. *Computamos todas las respuestas para la restricción definida por el usuario* $N\ /{=}\ s\ (count\ (from\ M))$ *usando el* $COISS(\mathcal{H}_{seq})$*-programa dado en el* Ejemplo 45 *y el árbol definicional dado en la* Figura 3.2. *Este ejemplo ilustra el uso de producciones para lograr el efecto de una evaluación* dirigida por demanda *con listas infinitas y el uso de árboles definicionales para asegurar la elección eficiente*

CS Resolución de Restricciones $\exists\overline{U}.\ P \square C \square S \square \sigma \Vdash_{\mathbf{CS}\{\chi\}} \exists\overline{Y}_i, \overline{U}.\ (P \square C)\sigma_i \square S_i \square \sigma\sigma_i$

Si $\chi = pvar(P)$, S no es χ-resuelto, $Solve^{\mathcal{D}}(S,\chi) = \bigvee_{i=1}^{k}(S_i \square \sigma_i)$, e $\overline{Y}_i$ son las nuevas variables introducidas por el resolutor en $S_i \square \sigma_i$, para todo $1 \leq i \leq k$.

AC Restricción Atómica $\exists\overline{U}.\ P \square\ p\overline{e}_n \to! \ t, C \square S \square \sigma \Vdash_{\mathbf{AC}} \exists\overline{X}_q, \overline{U}.\ \overline{e_q \to X_q}, P \square C \square\ p\overline{t}_n \to! \ t, S \square \sigma$

Si $p \in PF^n$, $p\overline{e}_n \to! \ t$ es una restricción, $\overline{X}_q$ son nuevas variables ($0 \leq q \leq n$ es el número de $e_i \notin Pat_\perp(\mathcal{U})$) tal que $t_i \equiv X_j$ $(0 \leq j \leq q)$ si $e_i \notin Pat_\perp(\mathcal{U})$ y $t_i \equiv e_i$ para cualquier otro caso para todo $1 \leq i \leq n$.

SF Fallo de Resolución $\exists\overline{U}.\ P \square C \square S \square \sigma \Vdash_{\mathbf{SF}\{\chi\}} \blacksquare$

si $\chi = pvar(P)$, S no es χ-resuelta, y $Solve^{\mathcal{D}}(S,\chi) = \blacklozenge$.

Figura 3.5: $CLNC(\mathcal{D})$-reglas para resolver restricciones y detección de errores.

de los redexes demandados.

$\square\ \underline{N\ /{=}\ s\ (count\ (from\ M))}\ \square\ \square\ \varepsilon \Vdash_{\mathbf{AC}}$ (restricción no primitiva)

$\exists L.\ \underline{s\ (count\ (from\ M)) \to L}\ \square\ \square\ N\ /{=}\ L\ \square \Vdash_{\mathbf{IM}\{\mathbf{L} \mapsto \mathbf{s\ K}\}}$ (L es necesario y demandado)

$\exists K.\ count\ (from\ M) \to K\ \square\ \square\ \underline{N\ /{=}\ s\ K}\ \square \Vdash_{\mathbf{CS}\{\mathbf{K}\}}$ (K es necesario pero no demandado)

Llegados a este punto, un resolutor de restricciones sobre $\mathcal{H}_{seq}$ (ver Ejemplo 37) da dos alternativas posibles: $Solve^{\mathcal{H}_{seq}}(\{N\ /{=}\ s\ K\}, \{K\}) = (\square\ \{N \mapsto 0\}) \vee (\{N'\ /{=}\ K\} \square \{N \mapsto s\ N'\})$, *y hay dos posibles continuaciones para el cómputo, donde el uso de árboles definicionales $\mathcal{T}_{from}$, $\mathcal{T}_{count}$ y $\mathcal{T}_{aux}$ en producciones demandadas guía y evita elecciones* don't know *de reglas de programa y computos fallidos con respecto al cálculo $CLNC(\mathcal{D})$ [50]:*

$\exists K.\ \underline{count\ (from\ M) \to K}\ \square\ \square\ \square\ \{N\ \mapsto\ 0\} \Vdash_{\mathbf{EL}}$ (K ¡ahora es innecesario!)

$\square\ \square\ \square\ \{N\ \mapsto\ 0\}$ **respuesta computada:** $S_1 \square \sigma_1\ \equiv\ \square\ \{N\ \mapsto\ 0\}$

$\exists K, N'.\ \underline{count\ (from\ M) \to K}\ \square\ \square\ N'\ /{=}\ K\ \square\ \{N \mapsto s\ N'\} \Vdash_{\mathbf{DT}}$ (K ¡Ahora es demandada!)

$\exists K, N'.\ \underline{<count\ (from\ M),\ \mathcal{T}_{count}> \to K}\ \square\ \square\ N'\ /{=}\ K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DN}}$

$\exists K', K, N'.\ \underline{from\ M \to K', <count\ K',\ \mathcal{T}_{count}> \to K}\ \square\ \square\ N'\ /{=}\ K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DT}}$

$\exists K', K, N'.\ \underline{<from\ M,\ \mathcal{T}_{from}> \to K', <count\ K', \mathcal{T}_{count}> \to K}\ \square\ \square$ (K' es demandada)

$N'\ /{=}\ K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{RRA}}$ (aplicación de la regla 'from')

$\exists K', K, N'.\ \underline{[M|from\ (s\ M)] \to K', <count\ K',\ \mathcal{T}_{count}> \to K}\ \square\ \square$

$N'\ /{=}\ K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash^{3}_{\mathbf{IM}\{\mathbf{K'} \mapsto [\mathbf{A}|\mathbf{As}]\},\ \mathbf{SS}\{\mathbf{A} \mapsto \mathbf{M}\},\ \mathbf{CSS}}$

$\exists As, K, N'.\ \underline{from\ (s\ M) \to As},\ <count\ [M|As],\ \mathcal{T}_{count\ [\cdot|\cdot]} >\!\to K\ \ \square\ \square$
$N' /= K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DT}}$ (As es demandada por el árbol definicional)

$\exists As, K, N'.\ \underline{<from\ (s\ M),\ \mathcal{T}_{from} >\!\to As},\ <count\ [M|As],\ \mathcal{T}_{count\ [\cdot|\cdot]} >\!\to K\ \ \square\ \square$
$N' /= K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{RRA}}$ (aplicación de la regla 'from')

$\exists As, K, N'.\ \underline{[s\ M|from\ (s\ (s\ M))] \to As},\ \underline{<count\ [M|As],\ \mathcal{T}_{count\ [\cdot|\cdot]} >\!\to K}\ \ \square\ \square$
$N'\ /= K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash^{3}_{\mathbf{IM\{As\ \mapsto\ [B|Bs]\},\ SS\{B\ \mapsto\ s\ M\},\ CSS}}$

$\exists Bs, K, N'.\ from\ (s\ (s\ M)) \to Bs,\ \underline{<count\ [M, s\ M|Bs],\ \mathcal{T}_{count\ [\cdot,\cdot|\cdot]} >\!\to K}\ \ \square\ \square$
$N' /= K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{RRA}}$ (aplicación de la regla 'count')

$\exists R, N'', Bs, K, N'.\ from\ (s\ (s\ M)) \to Bs, \underline{aux\ R\ N'' \to K},\ count\ [s\ M|Bs] \to N''\ \square$
$seq\ M\ (s\ M) \to!\ R \square N' /= K \square\ \{N \mapsto s\ N'\} \Vdash_{\mathbf{DT}}$ (Bs, N'' no está demandada)

$\exists R, N'', Bs, K, N'.from\ (s\ (s\ M)) \to Bs,\ \underline{<aux\ R\ N'', \mathcal{T}_{aux} >\!\to K},\ count\ [s\ M|Bs] \to N''\ \square$
$seq\ M\ (s\ M) \to!\ R\ \square\ N'\ /= K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DI\{R\ \mapsto\ false\}}}$

$\exists N'', Bs, K, N'.\ from\ (s\ (s\ M)) \to Bs,\ \underline{<aux\ false\ N'',\ \mathcal{T}_{aux,false} >\!\to K},$
$count\ [s\ M|Bs] \to N''\ \square M/= s\ M \square N'/= K \square \{N \mapsto s\ N'\} \Vdash^{2}_{\mathbf{RRA,SS\{K\ \mapsto\ s\ 0\}}}$

$\exists N'', Bs, N'.\ \underline{from\ (s\ (s\ M)) \to Bs}, \underline{count\ [s\ M|Bs] \to N''}\ \square\ M\ /= s\ M\ \square$
$N'\ /= s\ 0\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash^{2}_{\mathbf{EL}}$ (N'' es innecesaria y entonces Bs es innecesaria!)

$\exists N'. \square \underline{M/= s\ M} \square N'/= s\ 0 \square \{N \mapsto s\ N'\} \Vdash^{2}_{\mathbf{AC,CS\{\}}}$ *(tiene dos posibilidades de nuevo)*
$\exists N'.\ \square\ \square\ N'\ /= s\ 0\ \square\ \{M\ \mapsto\ 0, N\ \mapsto\ s\ N'\}$
respuesta computada: $S_2 \square \sigma_2 \equiv N'\ /= s\ 0\ \square\ \{M\ \mapsto\ 0, N\ \mapsto\ s\ N'\}$
$\exists M', N'.\ \square\ \square\ M'\ /= M, N'\ /= s\ 0\ \square\ \{M\ \mapsto\ s\ M', N\ \mapsto\ s\ N'\}$
respuesta computada: $S_3 \square \sigma_3 \equiv M'\ /= M, N'\ /= s\ 0\ \square\ \{M\ \mapsto\ s\ M',\ N\ \mapsto\ s\ N'\}$

3.4.3 Propiedades del Cálculo $CDNC(\mathcal{D})$

En este apartado mostramos los resultados teóricos principales de esta sección relativos a la relación existente entre la lógica $CRWL(\mathcal{D})$ y el mecanismo de resolución de objetivos $CDNC(\mathcal{D})$, y a los cuales denominaremos *corrección* y *completitud* del cálculo $CDNC(\mathcal{D})$ con respecto a la semántica del esquema $CFLP(\mathcal{D})$. Para probar ambas propiedades usaremos en el Apéndice técnicas similares a las usadas para el $CLNC(\mathcal{D})$-cálculo presentado en [50]. Los siguientes resultados de corrección demuestran que las respuestas computadas para un objetivo G son, en efecto, respuestas correctas.

Teorema 53 (Corrección de $CDNC(\mathcal{D})$). *Si G_0 es un objetivo inicial y $G_0 \Vdash^{*}_{CDNC(\mathcal{D})} G_n$, donde $G_n \equiv \exists \overline{U}.\ \square\ \square\ S \square \sigma$ es un objetivo resuelto, entonces $S \square \sigma \in Ans_{\mathcal{P}}(G_0)$.*

La demostración de completitud de $CDNC(\mathcal{D})$ está basada en la siguiente idea: siempre que $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$ y G no está resuelto todavía, hay un número finito de elecciones locales para un primer paso de cómputo $G \Vdash G_j$ $(1 \leq j \leq l)$ tal que el nuevo objetivo G_j está "más cerca de ser resuelto" y "cubre todas las soluciones de $\Pi \square \theta$". El siguiente resultado de completitud revela que $CDNC(\mathcal{D})$ es *fuertemente completo*, es decir, la elección local de la regla de transformación de objetivos

aplicada a cada paso puede ser una elección *don't care*.

Teorema 54 (Completitud de $CDNC(\mathcal{D})$). *Sea G_0 un objetivo inicial admisible y $\Pi_0 \square\ \theta_0 \in Ans_{\mathcal{P}}(G_0)$ no trivial. Entonces existe un número finito de derivaciones que terminan en objetivos resueltos $G_0 \vdash^*_{CDNC(\mathcal{D})} G_i$ $(1 \leq i \leq k)$ tales que $Sol_{\mathcal{D}}(\Pi_0 \square\ \theta_0) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{P}}(G_i)$.*

Finalmente, desde el punto de vista de la eficiencia y de acuerdo con nuestro *Lema 51 de Demanda*, los árboles definicionales se usan en las producciomes demandadas para asegurar que solo se den pasos de estrechamiento necesarios, en la línea de [5, 7, 21]. Entonces, los cómputos en $CDNC(\mathcal{D})$ son en esencia derivaciones de estrechamiento necesario módulo elecciones no deterministas entre reglas de programa con restricciones y solapamiento.

3.4.4 Transformación de Programas

Para demostrar que los $COISS(\mathcal{D})$-programas tienen la misma que expresividad que los $CFLP(\mathcal{D})$-programas, proponemos un *algoritmo de transformación* inspirado en la transformación descrita en [48] para probar que cualquier $CFLP(\mathcal{D})$-programa puede transformarse en un $COISS(\mathcal{D})$-programa equivalente, preservando la semántica de la lógica de reescritura $CRWL(\mathcal{D})$. Vamos a empezar definiendo algunas nociones auxiliares.

Definición 55 (Posiciones Demandadas). *Sea $\mathcal{P}$ un $CFLP(\mathcal{D})$-programa. Un* patrón de llamada genérico *es cualquier patrón de llamada de la forma $f\overline{X}_n$, donde $\overline{X}_n$ son variables diferentes. Sea τ un patrón de llamada. Se dice que:*

1. *Una regla de programa $(l \to r \Leftarrow P \square C) \in \mathcal{P}$ es* ajustada *por τ si y solo si $\tau \preceq l$.*

2. *$p \in VPos(\tau)$ es* demandada por el lado izquierdo *l de una regla de programa $\mathcal{P}$, la cual es ajustada por τ si y solo si l tiene un símbolo pasivo en la posición p.*

3. *$p \in VPos(\tau)$ es* demandada *si y solo si p es demandada por el lado izquierdo de una regla de programa en $\mathcal{P}$ que es ajustada por τ.*

4. *$p \in VPos(\tau)$ es* uniformemente demandada *si y solo si p es demandada por el lado izquierdo de todas las reglas de programa en $\mathcal{P}$ que se ajusten con τ.*

Sea $\mathcal{P}$ un $CFLP(\mathcal{D})$-programa. El *algoritmo de transformación* es ejecutado por una función denominada $COIS$ (*Inductivamente Secuencial con Restricciones y Solapamiento*) sobre $\mathcal{D}$ que toma $\mathcal{P}$ y devuelve un $COISS(\mathcal{D})$-programa equivalente. Para transformar completamente los programas sólo necesitamos aplicar

este algoritmo a cada función definida $f \in DF^n$ del programa con la *llamada inicial* $COIS(\mathcal{P}) = \bigcup_{f \in DF} COIS(f\overline{X}_n, \mathcal{P}_f)$, donde $f\overline{X}_n$ es un patrón de llamada genérico para todo $f \in DF^n$ y $\mathcal{P}_f$ es el subconjunto de $\mathcal{P}$ formado por las reglas definiendo f (obviamente, $\mathcal{P} = \bigcup_{f \in DF} \mathcal{P}_f$). Las llamadas recursivas tendrán la forma $COIS(\tau, \mathcal{P})$, donde τ es un patrón de llamada y $\mathcal{P}$ es un conjunto de reglas de $CFLP(\mathcal{D})$-programa que son ajustadas por τ. Distinguimos los siguientes casos:

1. *Alguna posición en $VPos(\tau)$ es uniformemente demandada.*

   Sea $p \in VPos(\tau)$ la posición más a la izquierda y sea X la variable en la posición p en τ. Sean $h_1, \ldots, h_m$ los símbolos pasivos que aparecen en la posición p en el lado izquierdo de las reglas del programa $\mathcal{P}$ ajustados por τ. Definimos $COIS(\tau, \mathcal{P}) = \bigcup_{1 \leq i \leq m} COIS(\tau_i, \mathcal{P}_i)$, donde:

   - $\tau_i = \tau\{X \mapsto h_i\overline{X}_{r_i}\}$ con $\overline{X}_{r_i}$ variables frescas.
   - $\mathcal{P}_i = \{(l \to r \Leftarrow P \,\square\, C) \in \mathcal{P} \mid l \succeq \tau_i\}$.

   Entonces $\mathcal{T}_\tau$ es de la forma $\underline{case}(\tau, X, [\mathcal{T}_{\tau_1}, \ldots, \mathcal{T}_{\tau_m}])$, donde los árboles $\mathcal{T}_{\tau_i}$ son creados recursivamente.

2. *Ninguna posición de $VPos(\tau)$ es demandada.*

   Estamos en el caso en el que todos los lados izquierdos de las reglas de $\mathcal{P}$ que son ajustadas por τ son variantes de τ y pueden definirse como $COIS(\tau, \mathcal{P}) = \mathcal{P}$. Además, $\mathcal{T}_\tau$ tiene la estructura $\underline{rule}(\tau \to r_1 \Leftarrow P_1 \,\square\, C_1 | \ldots | r_m \Leftarrow P_m \,\square\, C_m)$.

3. *Alguna posición en $VPos(\tau)$ es demandada, pero ninguna es uniformemente demandada.*

   Sea $p \in VPos(\tau)$ la posición más a la izquierda. Tomamos $\mathcal{P}_p^+ = \{R \in \mathcal{P} \mid$ el lado izquierdo de R que demanda p $\}$ y $\mathcal{P}_p^- = \mathcal{P} \backslash \mathcal{P}_p^+$. Consideramos dos nuevos símbolos de función, f_1, f_2, con la misma aridad que f (el símbolo de función definida en la raíz de τ), y definimos $COIS(\tau, \mathcal{P}) = COIS(\tau_1, \mathcal{P}_1) \cup COIS(\tau_2, \mathcal{P}_2) \cup \{\tau \to \tau_1, \tau \to \tau_2\}$, donde:

   - τ_1 (τ_2 respectivamente) es el nuevo patrón de llamada construido cambiando las apariciones de f en la raíz de τ por el nuevo símbolo de función f_1 (f_2 respectivamente).
   - $\mathcal{P}_1$ ($\mathcal{P}_2$ respectivamente) es el conjunto de programas construidos a partir de $\mathcal{P}_p^+$ ($\mathcal{P}_p^-$ respectivamente) cambiando la aparición de f en el lado izquierdo de sus reglas de programa por el nuevo símbolo de función f_1 (f_2 respectivamente).

El árbol definicional $\mathcal{T}_\tau$ tiene la forma $\underline{rule}(\tau \to \tau_1 \mid \tau_2)$ donde $\mathcal{T}_{\tau_1}$ y $\mathcal{T}_{\tau_2}$ son creados recursivamente.

Ejemplo 56. *Vamos a ilustrar con un ejemplo la aplicación del algoritmo de transformación. Consideremos el siguiente $CFLP(\mathcal{H}_{seq})$-programa $\mathcal{P}$ sobre el conjunto de los números enteros cuyas reglas definen una función inductivamente secuencial sin restricciones "check_list":*

$$
\begin{array}{llcll}
check\_list & [\,] & \to & 0 & \\
check\_list & [X|Xs] & \to & 1 & \Leftarrow \quad X \mathrel{/=} 1 \\
check\_list & [X|Xs] & \to & 2 & \Leftarrow \quad X \mathrel{/=} 0 \\
check\_list & [X,Y|Zs] & \to & 3 & \Leftarrow \quad X \mathrel{/=} Y
\end{array}
$$

Obtenemos un $COISS(\mathcal{D})$-programa equivalente $\mathcal{P}'$ mediante la aplicación del algoritmo de transformación:

$\mathcal{P}' = COIS(\mathcal{P}) = COIS(check\_list\ L, \mathcal{P}) =$

L está en una posición uniformemente demandada. Consideramos:

$\mathcal{P}_1 = \{check\_list\ [\,] \to 0\}$
$\mathcal{P}_2 = \mathcal{P} \setminus \mathcal{P}_1$

$= COIS(check\_list\ [\,], \mathcal{P}_1) \cup COIS(check\_list\ [X|Xs], \mathcal{P}_2) =$

- $COIS(check\_list\ [\,], \mathcal{P}_1) = \mathcal{P}_1$, *porque ninguna posición es demandada.*
- $COIS(check\_list\ [X|Xs], \mathcal{P}_2) =$

Xs está en una posición demandada, pero no uniformemente. Consideramos:

$\mathcal{P}_2^+ = \{check\_list\ [X,Y|Zs] \ \to\ 3 \Leftarrow\ X \mathrel{/=} Y\}$
$\mathcal{P}_2^- = \mathcal{P}_2 \setminus \mathcal{P}_2^+$

$\mathcal{P}_{21} = \{check\_list_1\ [X,Y|Zs] \ \to\ 3 \Leftarrow\ X \mathrel{/=} Y\}$
$\mathcal{P}_{22} = \{check\_list_2\ [X|Xs] \ \to\ 1 \Leftarrow\ X \mathrel{/=} 1,$
$\quad check\_list_2\ [X|Xs] \ \to\ 2 \Leftarrow\ X \mathrel{/=} 0\}$

$= COIS(check\_list_1\ [X|Xs], \mathcal{P}_{21}) \cup COIS(check\_list_2\ [X|Xs], \mathcal{P}_{22})$
$\cup\ \{check\_list\ [X|Xs] \ \to\ check\_list_1\ [X|Xs],$
$\quad check\_list\ [X|Xs] \ \to\ check\_list_2\ [X|Xs]\} =$

- $COIS(check\_list_1\ [X|Xs], \mathcal{P}_{21}) = COIS(check\_list_1\ [X,Y|\ Zs], \mathcal{P}_{21}) = \mathcal{P}_{21}$, *porque Xs está en una posición uniformemente demandada y X, Y, Zs no son demandadas.*

- $COIS(check\_list_2\ [X|Xs], \mathcal{P}_{22}) = \mathcal{P}_{22}$, *porque ninguna posición es demandada.*

Obtenemos $\mathcal{P}' = \mathcal{P}_1 \cup \mathcal{P}_{21} \cup \mathcal{P}_{22} \cup \{check\_list\ [X|Xs] \rightarrow check\_list_1\ [X|Xs]$, $check\_list\ [X|Xs] \rightarrow check\_list_2\ [X|Xs]\}$.

Finalmente, probamos que el algoritmo de transformación preserva la semántica de los programas, esto es: dado un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ y un $COISS(\mathcal{D})$-programa $\mathcal{P}'$ que es resultado de transformar $\mathcal{P}$, entonces las mismas c-sentencias son derivables en $CRWL(\mathcal{D})$ usando $\mathcal{P}$ y $\mathcal{P}'$.

Teorema 57 (**Propiedades de** $COIS$**).** *Para cualquier* $CFLP(\mathcal{D})$*-programa* $\mathcal{P}$ *dado,* $COIS(\mathcal{P})$ *siempre termina y devuelve otro* $CFLP(\mathcal{D})$*-programa* $\mathcal{P}'$ *tal que* $\mathcal{P}'$ *es un* $COISS(\mathcal{D})$ *y para cualquier c-sentencia* φ*:* $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ *si y solo si* $\mathcal{P}' \vdash_{\mathcal{D}} \varphi$.

El lector puede encontrar la demostración de este teorema en el Apéndice A del trabajo.

Capítulo 4

Programación Lógico Funcional Concurrente con Restricciones

En el capítulo anterior presentábamos el marco de programación lógico funcional con restricciones $CFLP(\mathcal{D})$ junto a la lógica de reescritura $CRWL(\mathcal{D})$, el cual nos aporta una semántica denotacional y operacional para programas y objetivos. Presentamos también la clase de los $COISS(\mathcal{D})$-programas junto con la definición de árbol definicional, que nos proporciona una forma eficiente de controlar los cómputos. Por último, vimos un algoritmo de transformación de $CFLP(\mathcal{D})$-programas a $COISS(\mathcal{D})$-programas, sobre los que podemos aplicar el cálculo de transformación de objetivos $CDNC(\mathcal{D})$. En este capítulo haremos uso de toda la base teórica aportada en los capítulos previos para presentar un marco $CFLP(\mathcal{D})$ concurrente. Tomaremos como base la definición de árbol definicional para la integración de residuación y cómputos concurrentes que se da en [33], dado que tiene en cuenta en qué momento una variable demandada permite avanzar el cómputo o lo detiene, obligando al objetivo a esperar a que las variables tomen un valor concreto. Una vez adaptada esta definición de árbol definicional al contexto del esquema $CFLP(\mathcal{D})$, podremos interpretar los $COISS(\mathcal{D})$-programas de forma concurrente mediante el etiquetado de las ramas, de forma que podamos controlar qué procesos sean los que le dan el valor a una variable mediante estrechamiento o resolución de restricciones, y qué procesos han de esperar a que dichas variables tomen un valor concreto (dado por otro proceso). Nuestra aportación será una semántica operacional concurrente correcta y completa para esta nueva versión del esquema $CFLP(\mathcal{D})$.

4.1 El Esquema de Programación Lógico Funcional Concurrente con Restricciones

En esta sección vamos a presentar la parte más novedosa de nuestro trabajo, donde añadimos concurrencia al esquema $CFLP(\mathcal{D})$ presentado en el Capítulo 3. Vamos

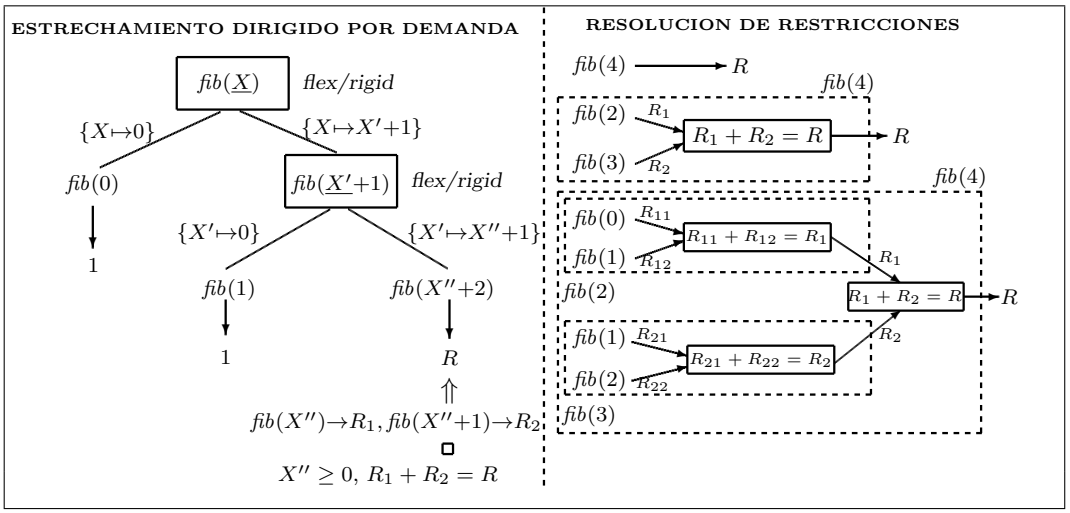

Figura 4.1: Concurrencia y paralelismo entre estrechamiento dirigido por demanda y resolución de restricciones.

a motivar en primer lugar, mediante un ejemplo concreto en el dominio finito $\mathcal{FD}$ de restricciones sobre números enteros, la forma en la que se combinan concurrentemente el estrechamiento demandado y la resolución de restricciones dentro del esquema $CFLP(\mathcal{D})$ y su implementación en $\mathcal{TOY}$.

Ejemplo 58. *Consideramos un programa cuyas reglas de reescritura permitan generar los términos de la sucesión de Fibonacci visto en el* Ejemplo 7 *del* Capítulo 2. *Observamos que la tercera regla del programa es una regla de reescritura con restricciones, ya que su parte derecha está condicionada mediante deficiones locales o auxiliares separadas de restricciones sobre el dominio $\mathcal{FD}$.*

$fib :: int \to int$

$fib(0) \to 1$
$fib(1) \to 1$
$fib(X+2) \to Y \Leftarrow fib(X) \to Y_1, fib(X+1) \to Y_2 \;\square\; X \geq 0, Y_1 + Y_2 = Y$

Queremos resolver, a partir de las reglas de este programa, el objetivo "$fib(X) \leq 2$", que consiste en determinar los valores enteros de la variable X para los que su correspondiente término en la sucesión de Fibonacci no sea mayor que 2. Para ello, el sistema concurrente $\mathcal{TOY}(\mathcal{FD})$ sobre el que se ejecuta este objetivo e implementa la instancia $CFLP(\mathcal{FD})$ del esquema comienza por separar ($\square$) la evaluación de la llamada a la función $fib(X)$ de la evaluación de la restricción $R \leq 2$, introduciendo para ello una variable auxiliar de comunicación R entre ellos.

$fib(X) \rightarrow R \ \square \ R \leq 2$

A partir de este momento, el sistema intenta evaluar, de manera concurrente, ambas partes del objetivo lanzando dos procesos separados: la llamada a la función $fib(X) \rightarrow R$ *mediante estrechamiento demandado, y la parte de la restricción* $R \leq 2$ *mediante el resolutor sobre* $\mathcal{FD}$ *asociado al sistema* $\mathcal{TOY}(\mathcal{FD})$*. Ahora bien, con el fin de preservar las condiciones de admisibilidad del esquema [14, 23] vistas en la Subsección 3.4.1 así como para evitar la desincronización en la comunicación entre los procesos a través de la variable lógica* R*, el sistema* $\mathcal{TOY}(\mathcal{FD})$ *ha de proteger la evaluación de la variable* R *de la acción del resolutor, quedando así este segundo proceso suspendido a la espera de obtener más información sobre el valor de* R*. Así pues, el sistema pasa a evaluar la llamada a la función* $fib(X) \rightarrow R$ *introduciendo para ello un árbol definicional con la opción* "flex" *activada en todas sus variables de selección con el fin de obtener valores para ellas mediante estrechamiento demandado por parte del sistema. Esto da lugar a la evaluación paralela del objetivo mediante tres procesos, correspondientes a la aplicación paralela de cada una de las reglas del programa:*

(1) ***Primera regla:*** $fib(0) \rightarrow 1$.

$\langle fib(X), \underline{case}(fib(X), X, flex, \ldots)\rangle \rightarrow R \ \square \ R \leq 2 \Vdash$
$\langle fib(0), \underline{rule}(fib(0) \rightarrow 1)\rangle \rightarrow R \ \square \ R \leq 2, \{X \mapsto 0\} \Vdash$
$1 \rightarrow R \ \square \ R \leq 2, \{X \mapsto 0\}$

En este momento, el valor de R *ha sido calculado y el proceso asociado al resolutor despierta para chequear la restricción instanciada con ese valor y confirmar el cómputo de la primera solución del objetivo inicial:*

$\square \ 1 \leq 2, \{R \mapsto 1, X \mapsto 0\} \Rightarrow \theta_1 = \{X \mapsto 0\}$ *(primera solución computada)*

(2) ***Segunda regla:*** $fib(1) \rightarrow 1$

$\langle fib\,(X' + 1), \underline{case}\,(fib\,(X' + 1), X', flex, \ldots)\rangle \rightarrow R \ \square \ R \leq 2, \ \{X \mapsto X' + 1\} \Vdash$

$\langle fib\,(1), \underline{rule}\,(fib\,(1) \rightarrow 1)\rangle \rightarrow R \ \square \ R \leq 2, \ \{X' \mapsto 0, X \mapsto 1\} \Vdash$

$1 \rightarrow R \ \square \ R \leq 2, \ \{X' \mapsto 0, X \mapsto 1\} \Vdash$

$\square \ 1 \leq 2, \ \{X' \mapsto 0, R \mapsto 1, X \mapsto 1\} \Rightarrow \theta_2 = \{X \mapsto 1\}$ *(segunda solución computada)*

(3) **Tercera regla:** $\mathit{fib}\,(X+2) \to Y \Leftarrow \mathit{fib}\,(X) \to Y_1,\ \mathit{fib}\,(X+1) \to Y_2 \ \square\ X \geq 0,\ Y_1 + Y_2 = Y.$

$\langle \mathit{fib}\,(X'+2), \underline{\mathit{rule}}\,(\mathit{fib}\,(X'+2) \to R \Leftarrow \ldots)\rangle \to R \ \square\ R \leq 2,\ \{X \mapsto X'+2\} \vdash$
$\underline{\mathit{fib}\,(X') \to R_1},\ \underline{\mathit{fib}\,(X'+1) \to R_2} \ \square\ \underline{X' \geq 0,\ R_1 + R_2 = R,\ R \leq 2,\ \{X \mapsto X'+2\}}$

A partir de este momento, la extensión concurrente del sistema $\mathcal{TOY}(\mathcal{FD})$ evalúa paralelamente este objetivo en dos formas, según sea posible dirigir el flujo de comunicación de los procesos entre el mecanismo de estrechamiento demandado y el resolutor de restricciones, como se puede apreciar en la Figura 4.1:

(3.1) ***Del estrechamiento a la resolución de restricciones:*** *El sistema evalúa en procesos paralelos las llamadas a las funciones auxiliares $fib(X')$ y $fib(X'+1)$. Sus resultados se van a guardar en variables auxiliares R_1 y R_2, respectivamente, y se comunicarán los procesos mediante la compartición del almacén de restricciones $X' \geq 0, R_1+R_2 = R, R \leq 2, \{X \mapsto X'+2\}$. Para asegurar la consistencia del canal de comunicación, el sistema ha de proteger las variables R_1 y R_2 de la acción del resolutor. Además, el árbol definicional para la función "fib" conserva la opción "$flex$" para la variable de selección X' con el fin de producir sus valores mediante estrechamiento demandado. En consecuencia, el sistema también debe proteger la variable X' de la acción del resolutor. Esto hace que el proceso correspondiente al resolutor quede suspendido en esta primera alternativa, no pudiendo hacer nada por simplificar el almacén de restricciones, y dando así ventaja al mecanismo de estrechamiento demandado frente a la resolución de restricciones. Se evalúan entonces las dos llamadas a "fib" de manera paralela, dando lugar por estrechamiento demandado a los siguientes dos resultados:*

$\square\ 1 + R_2 = R, R \leq 2, \{X' \mapsto 0, R_1 \mapsto 1, X \mapsto 2\}$

$\square\ R_1 + 1 = R, R \leq 2, \{X' \mapsto 0, R_2 \mapsto 1, X \mapsto 2\}$

El sistema combina ahora ambos resultados, despertando así el proceso del resolutor para que termine de comprobar la validez de la solución obtenida mediante el cálculo de la variable auxiliar R:

$1 + 1 = R, R \leq 2 \ \square\ \{X' \mapsto 0, R_1 \mapsto 1, R2 \mapsto 1, X \mapsto 2\} \vdash$
$2 \leq 2 \ \square\ \{X' \mapsto 0, R_1 \mapsto 1, R_2 \mapsto 1, R \mapsto 2, X \mapsto 2\} \Rightarrow \theta_3 = \{X \mapsto 2\}$ *(tercera solución computada)*

(3.2) ***De la resolución de restricciones al estrechamiento:*** *Paralelamente, el sistema concurrente $\mathcal{TOY}(\mathcal{FD})$ puede evaluar el objetivo sacando una mayor ventaja al resolutor de $\mathcal{FD}$ de SICStus Prolog frente al mecanismo de estrechamiento demandado. Si bien las variables R_1 y R_2 han de permanecer obligatoriamente protegidas de la acción del resolutor para mantener las condiciones de admisibilidad del esquema CFLP($\mathcal{D}$), la variable entera X' se puede intentar vincular mediante etiquetado y enumeración a partir de la restricción $X' \geq 0$. Para ello, el sistema necesita proteger la variable X' de la acción del estrechamiento demandado, cambiando la opción "$flex$" de X' en el árbol definicional de la función "fib" por la opción "$rigid$", evitando así que la variable X' pueda tomar valores enteros mediante la aplicación del estrechamiento a partir de las reglas del programa. Así, los procesos del sistema que van a evaluar separadamente las llamadas de función $fib(X')$ y $fib(X'+1)$ permanecen suspendidos en espera de la acción del resolutor. De este modo, el resolutor $\mathcal{FD}$ permite resolver la restricción $X' \geq 0$ enumerando valores enteros positivos para X' y creando las sustituciones: $\{X' \mapsto 0\}$, $\{X' \mapsto 1\}$, $\{X' \mapsto 2\}$, etc. Para cada uno de estos valores, el sistema crea un proceso paralelo. Al sustituir cada uno de estos valores de X' en el resto del objetivo, los procesos que evalúan cada llamada a la función "fib" mediante estrechamiento demandado se despiertan, mientras que el proceso del resolutor que evalúa el almacén de restricciones restante $R_1 + R_2 = R, R \leq 2$ (ya sin X') vuelve a quedar suspendido en espera de conocer más información sobre R_1 y R_2. Por ejemplo, para la sustitución $\{X' \mapsto 0\}$, el sistema computa paralelamente los valores de $fib(0)$ y $fib(1)$:*

$fib(0) \to R_1, fib(1) \to R_2 \ \square \ R_1 + R_2 = R, R \leq 2, \{X' \mapsto 0, X \mapsto 2\} \vdash$
$1 \to R_1, 1 \to R_2 \ \square \ R_1 + R_2 = R, R \leq 2, \{X' \mapsto 0, X \mapsto 2\} \vdash$
$\square \ 1 + 1 = R, R \leq 2, \{X' \mapsto 0, R_1 \mapsto 1, R_2 \mapsto 1, X \mapsto 2\} \vdash$
$\square \ 2 \leq 2, \{X' \mapsto 0, R_1 \mapsto 1, R_2 \mapsto 1, R \mapsto 2, X \mapsto 2\} \Rightarrow \theta_3 = \{X \mapsto 2\}$
(tercera solución computada)

Para el resto de valores de X' computados por el resolutor, todos los procesos terminan en fallo (■), dando por finalizado el cómputo paralelo de soluciones para el objetivo inicial:

$fib(1) \to R_1, fib(2) \to R_2 \ \square \ R_1 + R_2 = R, R \leq 2, \{X' \mapsto 1, X \mapsto 3\} \vdash$
$1 \to R_1, 2 \to R_2 \ \square \ R_1 + R_2 = R, R \leq 2, \{X' \mapsto 1, X \mapsto 3\} \vdash$
$\square \ 1 + 2 = R, R \leq 2, \{X' \mapsto 1, R_1 \mapsto 1, R_2 \mapsto 2, X \mapsto 3\} \vdash$
$\square \ \underline{3 \leq 2}, \{X' \mapsto 0, R_1 \mapsto 1, R_2 \mapsto 1, R \mapsto 2, X \mapsto 2\} \vdash \blacksquare$
(ninguna solución computada)

4.1.1 Árboles Definicionales Concurrentes

Para nuestro cálculo hemos adaptado la definición de árbol definicional dada en la *Definición 46* con la definición de árbol definicional de [33]:

Definición 59 (**Árbol definicional concurrente con patrón π**). *$\mathcal{T}$ es un* árbol definicional concurrente con patrón π *si y solo si la profundidad de $\mathcal{T}$ es finita y se cumple alguno de los siguientes casos:*

- $\mathcal{T} \equiv \underline{rule}(\tau \to r_1 \Leftarrow P_1 \,\square\, C_1 \parallel \ldots \parallel r_m \Leftarrow P_m \,\square\, C_m)$, *donde* $\tau \to r_i \Leftarrow P_i \,\square\, C_i$ *para todo* $1 \leq i \leq m$ *es una variante de una regla de programa en* $\mathcal{P}$.
- $\mathcal{T} \equiv \underline{case}(\tau, X, rigid/flex, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$, *donde* X *es una variable en* τ, $h_1, \ldots, h_k$ $(k > 0)$ *son símbolos pasivos de* $\mathcal{P}$ *diferentes dos a dos, y para todo* $1 \leq i \leq k$, $\mathcal{T}_i$ *es un* $cDT(\mathcal{D})$ *con patrón de llamada* $\tau\sigma_i$, *donde* $\sigma_i = \{X \mapsto h_i\bar{t}_{m_i}\}$ *con* $\bar{t}_{m_i}$ *elementos primitivos o* $\overline{Y}_{m_i}$ *nuevas varibles distintas tales que* $h_i\bar{t}_{m_i} \in Pat(\mathcal{U})$.

Definición 60. Un árbol definicional concurrente de una función f de aridad n *es un árbol definicional concurrente $\mathcal{T}$ con patrón $f(X_1, \ldots, X_n)$, donde $X_1, \ldots, X_n$ son variables distintas, tales que para cada regla $l \to r$ con $l = f(t_1, \ldots, t_n)$ existe un nodo $\underline{rule}(l' \to r')$ en $\mathcal{T}$ con l variante de l'. A partir de ahora, escribiremos $pat(\mathcal{T})$ para el patrón de un árbol definicional concurrente $\mathcal{T}$, y cDT para el conjunto de todos los arboles definicionales concurrentes.*

Como se puede apreciar, la definición de árbol definicional concurrente es muy similar a la dada en el capítulo anterior. Las principales diferencias las da la interpretación concurrente que aporta nuestra semántica operacional, y podemos apreciarlas en el siguiente ejemplo.

Ejemplo 61. *Dado el siguiente programa escrito en el lenguaje $CFLP(\mathcal{H})$*

$f(a, X) \to true$
$f(X, b) \to true$

tenemos un posible árbol definicional que mostramos en la Figura 4.2 bajo la interpretación de [33], donde cada bifurcación corresponde con los caminos por los que puede optar el cómputo indeterminista. Determinamos que la primera rama es de tipo "$rigid$" (el valor de la variable ha de estar instanciado para continuar por esta rama) y la segunda rama es de tipo "$flex$" (el valor de la variable será instanciado durante el cómputo). Con la semántica operacional propuesta, en cada bifurcación indeterminista, cada elección genera un nuevo proceso que generará una solución, indicando si el proceso es de tipo "$rigid$" o "$flex$". Con nuestra defición no se puede construir el árbol de la Figura 4.2 pero si aplicamos el algoritmo de transformación del capítulo anterior, se obtienen los árboles equivalente representados en la Figura

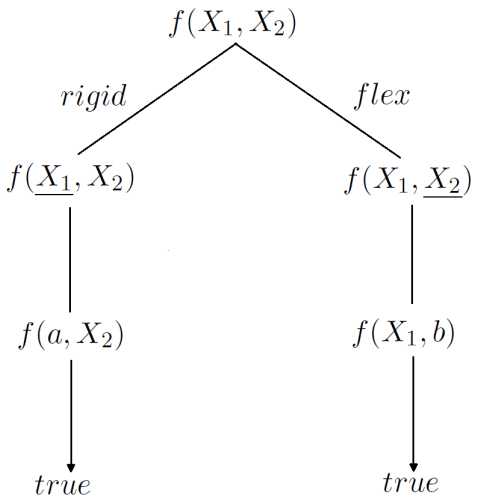

Figura 4.2: Árbol definicional del programa del Ejemplo 61.

4.3 y que definimos a continuación:

$\mathcal{T}_f \equiv \underline{rule}(f(X_1, X_2) \rightarrow f_1(X_1, X_2)||f_2(X_1, X_2))$
$\mathcal{T}_{f_1} = \underline{case}(f_1(X_1, X_2), X_1, rigid, [\underline{rule}(f_1(a, X_2) - > true)])$
$\mathcal{T}_{f_2} = \underline{case}(f_2(X_1, X_2), X_2, flex, [\underline{rule}(f_2(X_1, b) - > true)])$

Escribiremos $DVar_{\mathcal{D}}(G)$ (o más precisamente $DVar_{\mathcal{D}}(P \,\square\, S)$ o $DVar_{\mathcal{D}}(S)$) para denotar el conjunto de variables demandadas de el objetivo G, y $FVar(G)$, $RVar(G)$ (o más precisamente $FVar(P)$, $RVar(P)$) para el conjunto de variables flexibles y rígidas, respectivamente.

4.1.2 Estrechamiento Demandado con Árboles Definicionales Concurrentes con Restricciones y Solapamiento

El objetivo de esta sección es dar el conjunto de reglas de transformación $G\ G'$ (para objetivos G, G') que definan nuestra semántica operacional concurrente. La semántica presentada es una extensión del cálculo $CDNC(\mathcal{D})$, por lo que nos centraremos en aquellas reglas que han sido modificadas con respecto al nuevo cálculo y aquellas reglas que hemos anñadido con el fin de modelar la concurrencia. Escribiremos $G\,^* \|_{i=1}^{k} G_i$ para representar *derivaciones paralelas.*Al igual que en Capítulo 3, se trata de un conjunto de reglas de transformación de objetivos admisibles, y mantendremos la misma definición de objetivo admisible (ver *Subsección 3.4.1*).

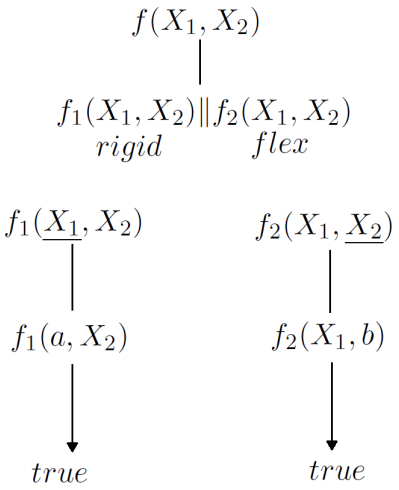

Figura 4.3: Árboles definicionales concurrentes del programa del Ejemplo 61.

Reglas Generales para Concurrencia en *CFLP*

A partir de un objetivo $G \equiv \exists \overline{U}.\, P \,\square\, C \,\square\, S \,\square\, \sigma$ se aplican las reglas del cálculo concurrente, indicado mediante la notación $\vdash\!\!\!\vdash^*$, hasta llegar a una de las siguientes tres situaciones: Se obtiene una respuesta $S \,\square\, \sigma$ identificada por un objetivo en forma resuelta $\exists \overline{U}.\, \square \,\square\, S \,\square\, \sigma$, se llega a un objetivo fallido $\blacksquare$, o bien la evaluación del objetivo queda suspendida, lo que se representa mediante el símbolo $\circlearrowleft$.

Cada vez que en un objetivo G se identifica la conjunción de dos partes atómicas susceptibles de ser evaluadas concurrentemente (por ejemplo, dos producciones, dos restricciones, una producción y una restricción, etc.), el sistema crea dos objetivos paralelos en procesos separados, G_1 y G_2, cada uno de ellos formado por una de las partes atómicas y toda aquella información necesaria para su correcta evaluación (variables producidas, demandadas, flexibles, rígidas, etc.). Ambos objetivos compartirán la misma parte resuelta del objetivo G (es decir, el mismo almacén de restricciones como veíamos en el Capítulo 2) para su sincronización, y se evaluarán paralelamente mediante la aplicación de las reglas del cálculo, lo que indicamos mediante la notación $G_1 \parallel G_2$. Para poder combinar adecuadamente las posibles respuestas obtenidas a partir de G_1 y de G_2, así como los casos en los que se queden suspendidos los procesos o los objetivos fallen, se utilizan las reglas generales de concurrencia en *CFLP* dadas en la Figura 4.4.

CR Combination Rules

$$\exists \overline{U}.\, P \square C \square S \square \sigma \vdash_{\mathbf{CR_1}} \left\{ \begin{array}{ll} \exists \overline{U}_1, \overline{U}_2.\, \square\ \square S_1\sigma_2, S_2\sigma_1 \square \sigma_1\sigma_2 & \text{si } \sigma_1, \sigma_2 \text{ se pueden componer} \\ \blacksquare & \text{en otro caso} \end{array} \right\}$$

$$\text{si } \exists \overline{U}.\, P \square C \square S \square \sigma \vdash^* \exists \overline{U}_1.\, \square\ \square S_1 \square \sigma_1 \parallel \exists \overline{U}_2.\, \square\ \square S_2 \square \sigma_2.$$

$$\exists \overline{U}.\, P \square C \square S \square \sigma \vdash_{\mathbf{CR_2}} \left\{ \begin{array}{ll} \exists \overline{U}_1, \overline{U}_2.\, P_1\sigma_2 \square C_1\sigma_2 \square S_1\sigma_2, S_2\sigma_1 \square \sigma_1\sigma_2 & \text{si } \sigma_1, \sigma_2 \text{ se pueden componer} \\ \circlearrowleft & \text{en otro caso} \end{array} \right\}$$

$$\text{si } \exists \overline{U}.\, P \square C \square S \square \sigma \vdash^* \parallel \left\{ \begin{array}{l} \exists \overline{U}_1.\, P_1 \square C_1 \square S_1 \square \sigma_1 \quad \vdash \quad \circlearrowleft \\ \exists \overline{U}_2.\, \square\ \square S_2 \square \sigma_2 \end{array} \right\}.$$

$\exists \overline{U}.\, P \square C \square S \square \sigma \vdash_{CR_3} \exists \overline{U}'.\, \square\ \square S' \square \sigma'$ si $\exists \overline{U}.\, P \square C \square S \square \sigma \vdash^* \exists \overline{U}'.\, \square\ \square S' \square \sigma' \parallel \blacksquare$.

$\exists \overline{U}.\, P \square C \square S \square \sigma \vdash_{CR_4} \circlearrowleft$ si $\exists \overline{U}.\, P \square C \square S \square \sigma \vdash^* \circlearrowleft \parallel \circlearrowleft$.

$\exists \overline{U}.\, P \square C \square S \square \sigma \vdash_{CR_5} \circlearrowleft$ si $\exists \overline{U}.\, P \square C \square S \square \sigma \vdash^* \blacksquare \parallel \circlearrowleft$.

$\exists \overline{U}.\, P \square C \square S \square \sigma \vdash_{CR_6} \blacksquare$ si $\exists \overline{U}.\, P \square C \square S \square \sigma \vdash^* \blacksquare \parallel \blacksquare$.

Figura 4.4: Reglas generales de concurrencia en *CFLP*: Combinación de respuestas.

Reglas para Concurrencia y Estrechamiento Demandado con Restricciones

Las reglas de transformación de objetivos correspondientes a producciones están diseñadas con la intención de modelar el comportamiento del estrechamiento dirigido por demanda mediante árboles definicionales concurrentes con restricciones y solapamiento, incluyendo funciones primitivas, funciones definidas y variables funcionales posiblemente de orden superior. Existen dos tipos de reglas, de acuerdo con los dos tipos posibles de producciones que pueden aparecer en un objetivo: reglas para suspensiones y reglas para producciones demandadas.

Las reglas correspondientes a suspensiones $e \to X$ se describen en la Figura 4.5 y se estructuran de acuerdo a la forma sintáctica de la expresión e.

EL Si la variable X no aparece en el resto del objetivo, la suspensión es innecesaria y puede ser eliminada. En el resto de reglas, supondremos que la variable X

EL ELimination

$\exists X, \overline{U}.\, e \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{EL}} \exists \overline{U}.\, P \square C \square S \square \sigma$ si $X \notin Var(P \square C \square S \square \sigma)$.

SS Simple Suspension

$\exists X, \overline{U}.\, t \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{SS}} \exists X, \overline{U}.\, (P \square C \square S)\sigma_0 \square \sigma\sigma_0$ si $t \in Pat(\mathcal{U})$, $\sigma_0 = \{X \mapsto t\}$.

IM IMitation

$$\exists X, \overline{U}.\, h\overline{e}_m \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{IM}} \left\{ \begin{array}{rr} \| \exists \overline{X}_m, X, \overline{U}.\, (\overline{e_m \to X_m}, P \square C \square S)\sigma_0 \square \sigma\sigma_0 & \\ & \text{si } X \in DVar_{\mathcal{D}}(P \square S) \\ \circlearrowleft & \text{si } X \notin DVar_{\mathcal{D}}(P \square S) \end{array} \right\}$$

si $h\overline{e}_m \notin Pat(\mathcal{U})$ es pasivo, y $\sigma_0 = \{X \mapsto h\overline{X}_m\}$ con $\overline{X}_m$ nuevas variables tales que $h\overline{X}_m \in Pat(\mathcal{U})$.

PF Primitive Function

$$\exists X, \overline{U}.\, p\overline{e}_n \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{PF}} \left\{ \begin{array}{rr} \| \exists \overline{X}_q, X, \overline{U}.\overline{e_q \to X_q}, P \square C \square p\bar{t}_n \to! X, S \square \sigma & \\ & \text{si } X \in DVar_{\mathcal{D}}(P \square S) \\ \circlearrowleft & \text{si } X \notin DVar_{\mathcal{D}}(P \square S) \end{array} \right\}$$

si $p \in PF^n$, y $\overline{X}_q$ son nuevas variables ($0 \le q \le n$ es el número de $e_i \notin Pat(\mathcal{U})$) s.t. $t_i \equiv X_j$ $(0 \le j \le q)$ si $e_i \notin Pat(\mathcal{U})$ y $t_i \equiv e_i$ en otro caso para cada $1 \le i \le n$.

DT Definitional Tree

$$\exists X, \overline{U}.\, f\overline{e}_n \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{DT_1}} \left\{ \begin{array}{rr} \exists X, \overline{U}. < f\overline{e}_n, \mathcal{T}_{f\overline{X}_n} > \to X, P \square C \square S \square \sigma & \\ & \text{si } X \in DVar_{\mathcal{D}}(P \square S) \\ \circlearrowleft & \text{si } X \notin DVar_{\mathcal{D}}(P \square S) \end{array} \right\}$$

$$\exists X, \overline{U}.\, f\overline{e}_n\overline{a}_k \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{DT_2}} \left\{ \begin{array}{rr} \exists X, X', \overline{U}. < f\overline{e}_n, \mathcal{T}_{f\overline{X}_n} > \to X', X'\overline{a}_k \to X, & \\ P \square C \square S \square \sigma & \text{si } X \in DVar_{\mathcal{D}}(P \square S) \\ \circlearrowleft & \text{si } X \notin DVar_{\mathcal{D}}(P \square S) \end{array} \right\}$$

si $f \in DF^n (k > 0)$, y tanto X' como todas las variables en $\mathcal{T}_{f\overline{X}_n}$ son nuevas variables.

FV Functional Variable

$$\exists X, \overline{U}.\, F\overline{e}_q \to X, P \square C \square S \square \sigma \Vdash_{\mathbf{FV}} \left\{ \begin{array}{rr} \|_h \exists \overline{X}_p, X, \overline{U}.\, (h\overline{X}_p\overline{e}_q \to X, P \square C \square S)\sigma_0 \square \sigma\sigma_0 & \\ & \text{si } F \notin PVar(P) \text{ and } X \in DVar_{\mathcal{D}}(P \square S) \\ \circlearrowleft & \text{en otro caso} \end{array} \right\}$$

si $q > 0$, $\sigma_0 = \{F \mapsto h\overline{X}_p\}$, y $\overline{X}_p$ son nuevas variables tales que $h\overline{X}_p \in Pat(\mathcal{U})$.

Figura 4.5: Reglas para estrechamiento dirigido por demanda concurrente: Suspensiones.

sí es necesaria, y por tanto, que aparece en alguna otra parte del objetivo.

SS Si e es un patrón t, es posible vincular la variable X a t y propagar este valor

al resto del objetivo mediante la creación de un vínculo de respuesta.

IM Si e no es un patrón sino una expresión pasiva $h\overline{e}_m$ y X es una variable demandada, se imita en X la estructura sintáctica de e reemplazando sus argumentos e_i mediante variables $\overline{X}_m$ y se propaga su nuevo valor. La evaluación de cada uno de sus argumentos e_i en variables X_i se puede paralelizar de manera cooperativa y después combinar ($\|$). Si la variable X no es demandada, el proceso que evalúa el objetivo queda suspendido ($\circlearrowleft$).

PF Si e comienza con un símbolo de función primitiva $p\overline{e}_n$ y X es demandada, se evalúan y combinan paralelamente de manera cooperativa sus argumentos e_i en las variables correspondientes X_i y se deposita la restricción primitiva $p\overline{X}_n \rightarrow! X$ en el almacén de restricciones compartido por los procesos. En caso de que la variable X no sea demandada, el proceso quedará suspendido.

DT Si e tiene un símbolo de función definida en la raíz y R es una variable demandada, solo esta transformaciones es aplicable, despertando la suspensión, decorando e con el $cDT(\mathcal{D})$ apropiado, e introduciendo una nueva producción demandada en el objetivo. En cualquier otro caso, el objetivo queda suspendido.

FV Finalmente, si e contiene una variable de orden superior F no producida y X es demandada, se evalúan en procesos paralelos no combinables cada posible vinculación de la variable F a un patrón $h\overline{X}_p$. En caso contrario, el proceso que evalúa el objetivo queda suspendido.

Las reglas de transformación de objetivos para producciones demandadas $< e, \mathcal{T} > \rightarrow R$ (ver Figura 4.6) codifican la estrategia de estrechamiento necesário guiado por el árbol definicional $\mathcal{T}$, de forma similar a [33]:

RRA Si $\mathcal{T}$ es un árbol de tipo *rule*, entonces puede aplicarse esta transformación en paralelo para cada una de las reglas de programa disponibles que ajusten con e, introduciendo las suspensiones y restricciones apropiadas en el nuevo objetivo de tal manera que se pueda asegurar una evaluación perezosa.

Si $\mathcal{T}$ es un árbol de tipo *case*, debe aplicarse una de las siguientes transformaciones: **CSS**, **DN** o **DI**, de acuerdo con el tipo de símbolo que aparece en la posición de la variable de distinción de casos de e. En cualquier otro caso, podemos llegar a fallo usando **CC** o puede suspenderse el cómputo usando **DP** o **DR** a la espera de un estado más avanzado del cómputo global.

CSS, **CC** Si h es un símbolo pasivo h_i, entonces **CSS** selecciona el subárbol apropiado $\mathcal{T}_i$ (si es posible; en otro caso, la regla **CC** fallaría).

DN Si h es un símbolo activo de función definida, o un símbolo activo de función primitiva, entonces la regla **DN** introduce una nueva suspensión demandada en el objetivo para evaluar $e|_p$.

DP, **DR**, **DI** Si h es una variable producida Y, el objetivo debe suspenderse usando la regla **DP** en espera de que su valor sea calculado por otro proceso paralelo. Si Y no es una variable producida, hay dos posibilidades. Si el nodo de bifurcación es *flex*, entonces **DI** selecciona cada subárbol $\mathcal{T}_i$ en paralelo, generando la asignación apropiada para Y. Si es *"rigid"*, entonces la regla de transformación **DR** suspende la evaluación de este proceso, emulando así el principio de cómputo basado en residuación de

Reglas para Concurrencia y Resolución de Restricciones Cooperativas

Nos interesa que nuestro cálculo sea capaz de manejar diferentes dominios de restricciones al mismo tiempo, de tal manera que los diferentes resolutores cooperen, e intercambien las restricciones que hayan fijado, mediante los *dominio de coordinación*. Un *dominio de coordinación* $\mathcal{C}$ [59] es un tipo de dominio de restricciones "híbrido" construido a partir de varios dominios de restricciones (como, por ejemplo, $\mathcal{H}$, $\mathcal{R}$, $\mathcal{FD}$, ...) para que cooperen. La construcción de dominios de coordinación incluye a los llamados *dominios mediadores* $\mathcal{M}$, cuyo propósito es el de proporcionar mecanismos de comunicación entre los dominios que componen el dominio de coordinación a través de puentes, proyecciones y otras operaciones. Vamos a tomar los dominios componentes $\mathcal{H}$, $\mathcal{R}$ y $\mathcal{FD}$, equipados con resolutores de restricciones, de tal manera que los mecanismos de comunicación beneficien a los resolutores. Explicaremos brevemente la construcción de este *dominio de coordinación* $\mathcal{C}$, representado como la suma $\mathcal{C} = \mathcal{M} \oplus \mathcal{H} \oplus \mathcal{FD} \oplus \mathcal{R}$.

Matemáticamente, la construcción del dominio de coordinación $\mathcal{C}$ depende de un dominio algebraico de restricciones combinado $\mathcal{FD} \oplus \mathcal{R}$ el cual representa la *suma amalgamada* de los dos dominios algebraicos combinables $\mathcal{FD}$ y $\mathcal{R}$. En este caso, las condiciones de combinación determinan que el único símbolo de función primitiva permitida permitida para pertenecer a $\mathcal{FD}$ y $\mathcal{R}$ es la igualdad estricta ==. Como consecuencia, la suma amalgamada $\mathcal{H} \oplus \mathcal{FD} \oplus \mathcal{R}$ siempre es posible; esto compone un dominio de restricciones que puede beneficiarse del resolutor $\mathcal{H}$. Sin embargo, son necesarios *dominios mediadores* para construir una suma más interesante para la cooperación algebráica hecha a medida de las comunicaciones entre los dominios puros $\mathcal{FD}$ y $\mathcal{R}$.

El dominio mediador $\mathcal{M}$ sirve como base para comunicaciones entre $\mathcal{FD}$ y $\mathcal{R}$, incluyendo la proyección de restricciones $\mathcal{R}$ al resolutor $\mathcal{FD}$ (y viceversa) mediante el uso de puentes, la especialización de restricciones $\mathcal{H}$ a restricciones de los dominios

RRA Rewrite Rule Application

$$\exists R, \overline{U}.\ < e, \underline{rule}(\tau \to r_1 \Leftarrow P_1 \square C_1 \parallel \ldots \parallel r_k \Leftarrow P_k \square C_k) >\!\!\to R, P \square C \square S \square \sigma \Vdash_{\mathbf{RRA}}$$
$$\|_{i=1}^{k}\ \exists \overline{X}, R, \overline{U}.\ \sigma_f(R_1) \to R_1, \ldots, \sigma_f(R_m) \to R_m, r_i\sigma_c \to R, P_i\sigma_c, P \square C_i\sigma_c, C \square S \square \sigma\sigma_c$$

- $\sigma_0 = \sigma_c \uplus \sigma_f$ con $Dom(\sigma_0) = Var(\tau)$ y $\tau\sigma_0 = e$.
- $\sigma_c =_{def} \sigma\restriction_{Dom_c(\sigma_0)}$, donde $Dom_c(\sigma_0) = \{X \in Dom(\sigma_0) \mid \sigma_0(X) \in Pat(\mathcal{U})\}$.
- $\sigma_f =_{def} \sigma\restriction_{Dom_f(\sigma_0)}$, donde $Dom_f(\sigma_0) = \{X \in Dom(\sigma_0) \mid \sigma_0(X) \notin Pat(\mathcal{U})\} = \{R_1, \ldots, R_m\}$.
- $\overline{X} \equiv Var(\tau \to r_i \Leftarrow P_i \square C_i) \backslash Dom_c(\sigma_0)$.

CSS Case Selection

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, \textit{flex/rigid}, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\to R, P \square C \square S \square \sigma \Vdash_{\mathbf{CSS}} \exists R, \overline{U}.\ < e, \mathcal{T}_i >\!\!\to R, P \square C \square S \square \sigma$$

si $e|_{pos(X,\tau)} = h_i \ldots$, con $1 \leq i \leq k$ dado por e, donde h_i es el símbolo pasivo asociado a $\mathcal{T}_i$.

CC Case non-Cover

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, \textit{flex/rigid}, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\to R, P \square C \square S \square \sigma \Vdash_{\mathbf{CC}} \blacksquare$$

si $e|_{pos(X,\tau)} = h \ldots$ es un símbolo pasivo y $h \notin \{h_1, \ldots, h_k\}$, donde h_i es el símbolo pasivo asociado a $\mathcal{T}_i$ $(1 \leq i \leq k)$.

DN Demand Narrowing

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, \textit{flex/rigid}, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\to R, P \square C \square S \square \sigma \Vdash_{\mathbf{DN}} \exists R', R, \overline{U}.\ e|_{pos(X,\tau)} \to R', < e[R']_{pos(X,\tau)}, \underline{case}(\tau, X, \textit{flex/rigid}, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\to R, P \square C \square S \square \sigma$$

si $e|_{pos(X,\tau)} = g \ldots$ con $g \in FS$ activo (símbolo de función definida o primitiva) y R' una nueva variable.

DP Demand Produced Variable

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, \textit{flex/rigid}, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\to R, P \square C \square S \square \sigma \Vdash_{\mathbf{DP}} \circlearrowleft$$

si $e|_{pos(X,\tau)} = Y$, $Y \in PVar(P)$.

DI Demand Instantiation

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, \textit{flex}, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\to R, P \square C \square S \square \sigma \Vdash_{\mathbf{DI}} \|_{i=1}^{k}\ \exists \overline{Y}_{m_i}, R, \overline{U}.\ (< e, \mathcal{T}_i >\!\!\to R, P \square C \square S)\sigma_0 \square \sigma\sigma_0$$

si $e|_{pos(X,\tau)} = Y$, $Y \notin PVar(P)$, $\sigma_0 = \{Y \mapsto h_i\overline{Y}_{m_i}\}$ con h_i $(1 \leq i \leq k)$ el símbolo pasivo asociado a $\mathcal{T}_i$ y $\overline{Y}_{m_i}$ son nuevas variables.

DR Demand Residuation

$$\exists R, \overline{U}.\ < e, \underline{case}(\tau, X, \textit{rigid}, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\to R, P \square C \square S \square \sigma \Vdash_{\mathbf{DR}} \circlearrowleft$$

si $e|_{pos(X,\tau)} = Y$, $Y \notin PVar(P)$.

Figura 4.6: Reglas para estrechamiento dirigido por demanda concurrente: Producciones demandadas.

$\mathcal{FD}$ o $\mathcal{R}$, y algunos otros mecanismos especiales diseñados para procesar las restricciones mediadoras que aparecen en las computaciones funcionales y lógicas.

Las restricciones *puente* $X \rightleftharpoons RX$, con $\rightleftharpoons \ :: int \to real \to bool$, pueden usarse tanto para *vínculos* como para *proyecciones*. Vincular en $solver^{\mathcal{M}}$ consiste en instanciar una variable que esté en el extremo de un puente siempre que el otro extremo del puente tome un valor numérico equivalente y compatible. Una proyección es una operación más compleja, la cual infiere restricciones del almacén de $\mathcal{R}$ que estén disponibles en el almacén de $\mathcal{FD}$ (y viceversa) y los puentes relevantes disponibles en $\mathcal{M}$. Esto permite a cada resolutor beneficiarse de los cómputos realizados por los demás resolutores. Postulamos una función de proyección $proj^{\mathcal{FD}\to\mathcal{R}}$ tal que para cualquier conjunto $C_{\mathcal{FD}}$ de $\mathcal{FD}$-restricciones y cualquier conjunto finito M de restricciones puente, $proj^{\mathcal{FD}\to\mathcal{R}}(C_{\mathcal{FD}}, M)$ devuelve una disyunción finita $C_{\mathcal{R}}$ de $\mathcal{R}$-restricciones equivalentes ($proj^{\mathcal{R}\to\mathcal{FD}}$ se define de forma similar). A fin de maximizar las posibilidades para la proyección, postulamos una función $bridges^{\mathcal{FD}\to\mathcal{R}}$ tal que $bridges^{\mathcal{FD}\to\mathcal{R}}(C_{\mathcal{FD}}, M)$ devuelve un conjunto finito de restricciones puente M' a partir de las nuevas variables en $C_{\mathcal{FD}}$ ($bridges^{\mathcal{R}\to\mathcal{FD}}$ se define de forma similar).

Las reglas de transformación de objetivos sobre restricciones (ver Figura 4.7) estan diseñadas para combinar de forma concurrente restricciones (primitivas o definidas por el usuario) por medio de la acción paralela de resolutores cooperativos de restricciones.

AC Esta regla transforma restricciones no primitivas $p\bar{e}_n \to! \ t$ mediante la aplicación de la evaluación concurrente de sus argumentos $e_q \to X_q$ en una forma aplanada consistente en una conjunción de restricciones primitivas $p\bar{t}_n \to! \ t$ con nuevas variables existenciales.

CS, **SF** Para cualquier dominio de restricciones $\mathcal{D}$ aplicamos un *resolutor de restricciones* $Solver^{\mathcal{D}}(S, \chi)$ el cual puede reducir cualquier conjunción finita de restricciones primitivas S en una forma equivalente más sencilla, tomándo adecuadamente un conjunto de *variables críticas* $\chi =_{def} PVar(P) \cup FVar(P)$ que aparecen en S. Necesitamos que cualquier invocación al resolutor devuelva una disyunción paralela finita $\exists\overline{Y}_i.\, S_i \,\square\, \sigma_i$ de almacenes de restricciones cuantificados existencialmente. La regla **CS** describe las transformaciones paralelas posibles de un mismo objetivo por una invocación del resolutor para cada alternativa posible. Puesto que el valor de una variable crítica suspendida puede ser ahora necesario y activada por algunas soluciones de una alternativa paralela S_i por demanda de la variable (es decir, $DVar_{\mathcal{D}}(S_i) \cap \chi \neq \emptyset$) o activar una producción mediante la aplicación de una sustitución σ_i e instanciando una variable demandada rígida en P (es decir, $Dom(\sigma_i) \cap RVar(P) \neq \emptyset$) o irrelevante para alguna otra solución alternativa S_i (es decir, $DVar_{\mathcal{D}}(S_i) \cap \chi \neq \emptyset$), necesitamos que los resolutores posean la habilidad de computar una distinción de casos paralela discriminando tales situaciones. Para cualquier otra situación,

AC Atomic Constraint

$$\exists \overline{U}.\, P \square\ p\bar{e}_n \rightarrow! \ t, C \square S \square \sigma \Vdash_{\mathbf{AC}} \| \ \exists \overline{X}_q, \overline{U}.\, \overline{e_q \rightarrow X_q}, P \square C \square\ p\bar{t}_n \rightarrow! \ t, S \square \sigma$$

si $p \in PF^n$, $p\bar{e}_n \rightarrow! \ t$ es una restricción, $\overline{X}_q$ son nuevas variables ($0 \leq q \leq n$ es el número de $e_i \notin Pat(\mathcal{U})$) tal que $t_i \equiv X_j$ ($0 \leq j \leq q$) si $e_i \notin Pat(\mathcal{U})$ y $t_i \equiv e_i$ en cualquier otro caso para cada $1 \leq i \leq n$.

CS Constraint Solving

$$\exists \overline{U}.\, P \square C \square S \square \sigma \Vdash_{\mathbf{CS}\{\chi\}} \|_{i=1}^{k} \left\{ \begin{array}{cc} \exists \overline{Y}_i, \overline{U}.\, (P \square C)\sigma_i \square S_i \square \sigma\sigma_i & \text{si } Var(S_i) \cap \chi = \emptyset \text{ o} \\ & DVar_{\mathcal{D}}(S_i) \cap \chi \neq \emptyset \text{ o} \\ & Dom(\sigma_i) \cap RVar(P) \neq \emptyset \\ \circlearrowleft & \text{en otro caso} \end{array} \right\}$$

- $\chi = PVar(P) \cup FVar(P)$.
- S no está en forma χ-resuelta.
- $Solver^{\mathcal{D}}(S, \chi) = \bigvee_{i=1}^{k}(S_i \square \sigma_i)$.
- $\overline{Y}_i$ son las nuevas variables introducidas por el resolutor en $S_i \square \sigma_i$, para cada $1 \leq i \leq k$.

SF Solving Failure

$$\exists \overline{U}.\, P \square C \square S \square \sigma \Vdash_{\mathbf{SF}\{\chi\}} \blacksquare$$

- $\chi = PVar(P) \cup FVar(P)$.
- S no está en forma χ-resuelta.
- $Solver^{\mathcal{D}}(S, \chi) = \blacklozenge$.

PP Propagate Projections

$$\exists \overline{U}.\, P \square C \square S_{\mathcal{D}}, S_{\mathcal{M}}, S \square \sigma \Vdash_{\mathbf{PP}} \exists \overline{Y}, \overline{U}.\, P \square C \square S_{\mathcal{D}}, S_{\mathcal{D}'}, S_{\mathcal{M}}, S \square \sigma$$

- $Proj^{\mathcal{D} \rightarrow \mathcal{D}'}(S_{\mathcal{D}}, S_{\mathcal{M}}) = S_{\mathcal{D}'}$.
- $\overline{Y}$ son las nuevas variables introducidas por la función de proyección.

SB Set Bridges

$$\exists \overline{U}.\, P \square C \square S_{\mathcal{D}}, S_{\mathcal{M}}, S \square \sigma \Vdash_{\mathbf{SB}} \exists \overline{Y}, \overline{U}.\, P \square C \square S_{\mathcal{D}}, S_{\mathcal{M}}, S'_{\mathcal{M}}, S \square \sigma$$

- $Bridges^{\mathcal{D} \rightarrow \mathcal{D}'}(S_{\mathcal{D}}, S_{\mathcal{M}}) = S'_{\mathcal{M}}$.
- $\overline{Y}$ con las nuevas variables introducidas por la dunción constructora de puentes.

Figura 4.7: Reglas para concurrencia y resolución cooperativa de restricciones.

el correspondiente objetivo debe suspenderse. La regla de fallo **SF** se usa para detección de fallo en la resolución de restricciones.

Mediante los siguiente ejemplos podemos ver el funcionamiento del sistema de reglas concurrente.

Ejemplo 62. *Para el programa definido en el Ejemplo 61 lanzamos el siguiente objetivo admisible:* $\square\ f(X,X) == true\ \square\ X == a\ \square\ \epsilon$

A partir de él, salen dos posibles ramas de cómputo, una para la restricción definida $f(X,X) == true$ *y otra para resolver la restricción primitiva* $X == a$*. Primero veamos cómo evoluciona el cómputo encargado de la restricción definida.*

$\square\ f(X,X) == true\ \square\ X == a\ \square\ \epsilon \Vdash_{AC}$

$\exists R.f(X,X) \rightarrow R\ \square\ \square\ R == true, X == a\ \square\ \epsilon \Vdash_{DT}$

$\exists R.\langle f(X,X), \mathcal{T}_f\rangle \rightarrow R\ \square\ \square\ R == true, X == a\ \square\ \epsilon \Vdash_{DS}$

$\exists R.\langle f(X,X), \underline{case}(f(X_1,X_2), X_1, rigid, \underline{rule}(f(a,X_2) \rightarrow true))\rangle \rightarrow R\ \square\ \square$
$R == true, X == a\ \square\ \epsilon \Vdash_{\circlearrowleft}$

Vemos que se llega a un punto en el que el cómputo se detiene debido a que el argumento demandado X *es de tipo "rigid", por lo que el cómputo se bloquea al no estar instanciado a ningún valor y no ser producido (es decir, se bloquea a la espera de poder ser calculado más adelante por otra parte del objetivo). En concreto, mientras se suspende el bloque anterior, de manera concurrente esperamos que el cómputo continue evaluando la restricción primitiva, tal y como podemos ver a continuación:*

$\square\ f(X,X) == true\ \square\ X == a\ \square\ \epsilon \Vdash_{CS}$

$Solve^{\mathcal{H}}(X == a\ \square\ \emptyset) =\ \square\ \{X \mapsto a\}\ \square\ f(a,a) == true\ \square\ \square\ \{X \mapsto a\} \Vdash_{AC}$

$\exists R.f(a,a) \rightarrow R\ \square\ \square\ R == true\ \square\ \{X \mapsto a\} \Vdash_{DT}$

$\exists R.\langle f(a,a), \mathcal{T}_f\rangle \rightarrow R\ \square\ \square\ R == true\ \square\ \{X \mapsto a\}$

Como podemos observar, este proceso del cómputo ha llegado a instanciar la variable X*, por lo que el cómputo puede continuar por la otra rama que se había bloqueado y obtener así un resultado:*

$\exists R.\langle f(a,a), \underline{case}(f(X_1,X_2), X_1, rigid, \underline{rule}(f(a,X_2) \rightarrow true))\rangle \rightarrow R\ \square\ \square\ R == true\ \square$
$\{X \mapsto a\} \Vdash_{CSS}$

$\exists R.\langle f(a,a), \underline{rule}(f(a,X_2) \rightarrow true)\rangle \rightarrow R\ \square\ \square\ R == true\ \square\ \{X \mapsto a\} \Vdash_{RRA}$

$\exists R.true \rightarrow R \ \square \ \square \ R == true \ \square \ \{X \mapsto a\} \Vdash_{SS}$

$\square \ \square \ true == true \ \square \ \{X \mapsto a\} \Vdash_{AC}$

$\boxed{\square \ \square \ \square \ \{X \mapsto a\}}$

Ejemplo 63. *Dado el siguiente programa que describe el proceso de ordenación "quickSort"*

$quickSort([]) \rightarrow []$
$quickSort([X|Xs]) \rightarrow Ys \Leftarrow split(Xs, X) == (X_{S_1}, X_{S_2}), quickSort(X_{S_1}) == Y_{S_1},$
$quickSort(X_{S_2}) == Y_{S_2}, append(Y_{S_1}, [X|Y_{S_2}]) == Y_S$

$split([], X) \rightarrow ([], [])$
$split([Y|Ys], X) \rightarrow ([Y|Ys_1], Ys_2) \Leftarrow Y \leq X, split(Ys, X) == (Ys_1, Ys_2)$
$split([Y|Ys], X) \rightarrow (Ys_1, [Y|Ys_2]) \Leftarrow Y > X, split(Ys, X) == (Ys_1, Ys_2)$

$append([], Ys) \rightarrow Ys$
$append([X|Xs], Ys) \rightarrow [X|Zs] \Leftarrow append(Xs, Ys) \rightarrow Zs$

para el objetivo admisible $\square \ quickSort([5,2,7]) == L \ \square \ \square \ \epsilon$ *un posible cómputo es el siguiente:*

$\square \ quickSort([5,2,7]) == L \ \square \ \square \ \epsilon \Vdash_{AC}$
$\exists R.quickSort([5,2,7]) \rightarrow R \ \square \ \square \ R == L \ \square \ \epsilon \Vdash^*$
$\exists X_{S_1}, X_{S_2}, R, Y_{S_1}, Y_{S_2}.split([2,7],5) \rightarrow (X_{S_1}, X_{S_2}) \ \square \ \square \ R == L \ \square \ \epsilon \Vdash^*$
$\left.\begin{array}{l} quicksort(X_{S_1}) \rightarrow Y_{S_1}, \\ quicksort(X_{S_2}) \rightarrow Y_{S_2}, \\ append(Y_{S_1}, [5|Y_{S_2}]) \rightarrow R \end{array}\right\} \Vdash^* \circlearrowleft$

$\exists X_{S_1}, X_{S_2}, R, Y_{S_1}, Y_{S_2}.([2],[7]) \rightarrow (X_{S_1}, X_{S_2}) \ \square \ \square \ R == L \ \square \ \epsilon \Vdash_{SS}$
$\left.\begin{array}{l} quicksort(X_{S_1}) \rightarrow Y_{S_1}, \\ quicksort(X_{S_2}) \rightarrow Y_{S_2}, \\ append(Y_{S_1}, [5|Y_{S_2}]) \rightarrow R \end{array}\right\} \Vdash^* \circlearrowleft$

$\exists Y_{S_1}, Y_{S_2}, R.quickSort([2]) \rightarrow Y_{S_1}, quickSort([7]) \rightarrow Y_{S_2}, append(Y_{S_1}, [5|Y_{S_2}]) \rightarrow R \ \square \ \square \ R == L \ \square \ \epsilon \Vdash$

En este punto del cómputo, los procesos correspondientes a las producciones "append" y los correspondientes a las restricciones primitivas están bloqueados puesto que las variables Y_{s_1} Y_{s_2} *y* R *son demandadas, pero no han sido instanciadas a ningún valor. Por ello el cómputo solo puede seguir dos caminos, uno por cada proceso cooperativo "quickSort", que se ejecutan concurrentemente.*

$\exists Y_{S_1}.quickSort([2]) \rightarrow Y_{S_1}\ \square\ \square\ \square\ \epsilon \vdash^*_{RRA}$

$\exists X_{S_{11}}, X_{S_{21}}, Y_{S_1}, Y_{S_{11}}, Y_{S_{21}}.split([],2) \rightarrow (X_{S_{11}}, X_{S_{21}})\ \square\ \square\ \square\ \epsilon \vdash^*_{RRA}$
$\left.\begin{array}{l} quicksort(X_{S_{11}}) \rightarrow Y_{S_{11}}, \\ quicksort(X_{S_{21}}) \rightarrow Y_{S_{21}}, \\ append(Y_{S_{11}}, [2|Y_{S_{21}}]) \rightarrow Y_{S_1} \end{array}\right\} \vdash^* \circlearrowleft$

$\exists X_{S_{11}}, X_{S_{21}}, Y_{S_1}, Y_{S_{11}}, Y_{S_{21}}.([],[]) \rightarrow (X_{S_{11}}, X_{S_{21}})\ \square\ \square\ \square\ \epsilon \vdash^*_{RRA}$
$\left.\begin{array}{l} quicksort(X_{S_{11}}) \rightarrow Y_{S_{11}}, \\ quicksort(X_{S_{21}}) \rightarrow Y_{S_{21}}, \\ append(Y_{S_{11}}, [2|Y_{S_{21}}]) \rightarrow Y_{S_1} \end{array}\right\} \vdash^* \circlearrowleft$

$\exists Y_{S_{11}}, Y_{S_{21}}, Y_{S_1}.\underbrace{quickSort([]) \rightarrow Y_{S_{11}}}_{[]\rightarrow Y_{S_{11}} \quad \|}, \underbrace{quickSort([]) \rightarrow Y_{S_{21}}}_{[]\rightarrow Y_{S_{21}} \quad \|}, \underbrace{append(Y_{S_{11}}, [2|Y_{S_{21}}])}_{\vdash \circlearrowleft} \rightarrow$
$Y_{S_1}\ \square\ \square\ \square\ \epsilon \vdash^*_{SS}$

$\exists Y_{S_1}.append([],[2]) \rightarrow Y_{S_1}\ \square\ \square\ \square\ \epsilon \vdash^* \exists Y_{S_1}.\square\square\square\{Y_{S_1} \mapsto [2]\}$

En este bloque se realiza el proceso "quickSort" de la lista unitaria '$[2]$*'. Al ser una lista unitaria, los procesos "quickSort" de esta llamada no generan más llamadas recursivas, sino que devuelven un resultado, por lo que el "append" local se desbloquea y se da valor a la variable demandada* Y_{S_1}*.*

$\exists Y_{S_2}.quickSort([7]) \rightarrow Y_{S_2}\ \square\ \square\ \square\ \epsilon \vdash^*$

$\exists Y_{S_2}.[7] \rightarrow Y_{S_2}\ \square\ \square\ \square\ \epsilon \vdash_{SS}$

$\exists Y_{S_2}.\ \square\ \square\ \square\ \{Y_{S_2} \mapsto [7]\}$

Este bloque se computa de forma concurrente con el anterior, siendo el proceso correspondiente a la producción '$append([2],[5,7])' \rightarrow R$ *que se realiza de forma análoga al bloque anterior. Este bloque de computo instancia la variable demandada* Y_{S_2} *al valor '*$[7]$*', lo que permite continuar al cómputo principal.*

$\exists R.append([2],[5,7]) \rightarrow R\ \square\ \square\ R == L\ \square\ \epsilon \vdash$

$\exists Zs, R.append([],[5,7]) \rightarrow Zs, [2|Zs] \rightarrow R\ \square\ \square\ R == L\ \square\ \epsilon \vdash$

$\exists Zs.[5,7] \rightarrow Zs\ \square\ \square\ [2|Zs] == L\ \square\ \epsilon \vdash$

$\square\ \square\ [2,5,7] == L\ \square\ \epsilon \vdash_{CS}$

$\square\ \square\ \square\ \{L \mapsto [2,5,7]\}$

Como ahora las variables Y_{S_1} e Y_{S_2} han sido instanciadas a los valores '[2]' y '[7]' respectivamente, el proceso correspondiente a la producción "append" que había quedado suspendido se desbloquea y da valor a la variable R, que a su vez es demandada por la restricción primitiva $R == L$, y aplicando la regla del resolutor de restricciones obtenemos la solución '$[2,5,7]$'.

4.1.3 Propiedades de la Semántica Operacional

Concluimos esta sección fijando un resultado teórico que nos permite garantizar la corrección y la completitud de las derivaciones paralelas en $CFLP(\mathcal{D})$.

Teorema 64 (Corrección y Completitud). *Sea G un objetivo inicial y $S \square \sigma$ una respuesta de G.*

(a) **Correción:** *Si $G \Vdash^* \|_{i=1}^k G_i$ es una derivavición paralela a partir de G de un número finito de objetivos G_i, para cada $G_i \equiv \exists \overline{U}.\ \square\ \square\ S_i \square \sigma_i$ un objetivo en forma resuelta, $S_i \square \sigma_i$ es una respuesta de G.*

(b) **Completitud:** *Existe una derivación paralela $G \Vdash^* \|_{i=1}^k G_i$ que termina en un número finito de objetivos resueltos $G_i \equiv \exists \overline{U}_i.\ \square\ \square\ S_i \square \sigma_i$ que cubren todas las soluciones de la respuesta inicial $S \square \sigma$.*

La demostración de este teorema se basa en la demostración de las propiedades del cálculo $CDNC(\mathcal{D})$ recopiladas en el Apéndice A, donde ahora las elecciones indeterministas han de interpretarse como derivaciones paralelas de procesos de resolución de objetivos.

Capítulo 5

Conclusiones y Trabajo Futuro

En este último capítulo del trabajo resumimos las principales aportaciones de nuestro trabajo, así como las principales lineas de investigación que quedan abiertas.

5.1 Conclusiones

El objetivo principal de este trabajo ha sido el de presentar una extensión concurrente para lenguajes de programación lógico funcionales con restricciones que permita modelar aplicaciones declarativas con restricciones donde la concurrencia y la resolución cooperativa de restricciones esté presente. Las principales aportaciones de este trabajo son las siguientes:

- Hemos ofrecido un estado del arte revisado y uniformado de las distintas propuestas existentes para la integración de concurrencia en el esquema de programación lógica con restricciones $CLP(\mathcal{C})$. Hemos mostrado cómo se han integrado las principales características de la programación lógica, como el indeterminismo y la unificación de variables lógicas sobre un sistema de restricciones $\mathcal{C}$, las cuales aportan una mayor expresividad y permiten resolver una gran cantidad de problemas de forma más eficiente, mostrando el paralelismo inherente que existe entre los cómputos de los programas lógicos, la resolución de restricciones y las redes de procesos.

- Hemos mostrado como integrar las principales características de la programación funcional, como la evaluación perezosa y el orden superior, al esquema de programación lógica con restricciones, ofreciendo una versión uniformada de la semántica operacional basada en estrechamiento dirigido por demanda mediante árboles definicionales con restricciones y solapamiento que sirve de fundamento al esquema de programación lógico funcional con restricciones $CFLP(\mathcal{D})$.

- Hemos integrado un modelo de cómputo concurrente en el esquema $CFLP(\mathcal{D})$. El conjunto de reglas de transformación que hemos aportado proporcionan un modelo operacional coherente que es correcto y completo respecto a la semántica $CRWL(\mathcal{D})$. Esta versión extendida del esquema nos permite combinar eficientemente los principios operacionales más importantes del paradigma de la programación declarativa con restricciones, como lo son el estrechamiento dirigido por demanda y los mecanismos de residuación guiados por árboles definicionales concurrentes con restricciones y solapamiento para añadir la posibilidad de realizar cómputos concurrentes y perezosos con restricciones.

5.2 Trabajo Futuro

Finalizamos este trabajo citando algunas de las principales líneas de trabajo que quedan abiertas después de la realización de esta memoria:

- Desde el punto de vista teórico, es necesario detallar las demostraciones de los resultados de corrección y completitud expuestos al final del Capítulo 4. Estas demostraciones tienen una estructura muy similar a las demostraciones de las propiedades del cálculo $CDNC(\mathcal{D})$, interpretando ahora las elecciones indeterministas como derivaciones paralelas entre objetivos.

- La principal línea de trabajo que queda abierta es la de adaptar el modelo de cómputo concurrente propuesto al sistema $\mathcal{TOY}(\mathcal{FD})$ desarrollado por el Grupo de Programación Declarativa (GPD) de la Universidad Complutense de Madrid. Actualmente ya es bastante común encontrarse con equipos con procesadores de doble núcleo, cuatro núcleos y superior, por lo que es posible ejecutar de forma simultánea varios procesos concurrentemente. Así pues, integrando nuestro modelo concurrente de cómputo con el sistema $\mathcal{TOY}(\mathcal{FD})$, se pueden conseguir mejoras en eficiencia notables al dividir el peso del cómputo en varios procesadores. El sistema $\mathcal{TOY}(\mathcal{FD})$ es un interprete para Prolog de $CFLP(\mathcal{D})$-programas construido sobre la distribución SICStus Prolog [2]. El sistema SICStus permite mezclar el lenguaje Prolog con C [1] y Java [42], lenguajes que poseen herramientas de creación y manejo de procesos e hilos. Por razones de eficiencia se recomienda el lenguaje C para una implementación para sistemas operativos UNIX. SICStus prolog por lo que nuestra propuesta sería la siguiente:

  - Cuando la evaluación de un objetivo llega a un punto de bifurcación debido al indeterminismo, generaríamos tantos procesos independientes como procesos, y el proceso padre quedaría suspendido a la espera de recopilar los resultados obtenidos por los cómputos de sus hijos.

- Para interpretar la concurrencia dentro de un mismo objetivo, dentro de nuestro modelo hablabamos de procesos que cooperan, por ello, por simplicidad proponemos que los "procesos" que cooperan sean implementados por hilos de ejecución. Cada una de las producciones y restricciones, en vez de generar un nuevo proceso independiente, genera un hilo, por lo que ahorramos en tiempo debido al coste de paso de mensajes, ya que la comunicación se realizaría por medio de paso de mensajes.

Además, habría que tener en cuenta detalles de implementación como control sobre las secciones críticas, detección de "*deadlocks*", etc.

- Una vez implementado el sistema $\mathcal{TOY}(\mathcal{FD})$ concurrente, realizar una implementación más general que permita la integración de resolutores de restricciones cooperativos.

Apéndices

Apéndice A

Demostraciones

En este Apéndice damos las demostraciones de los resultados principales expuestos en este trabajo, los cuales han sido separados del texto principal para facilitar su comprensión y legibilidad.

A.1 Demostraciones del Capítulo 2

Proposición (**Propiedades de los sistemas de restricciones con igualdad).** *Cualquier sistema de restricciones con igualdad, satisface las propiedades de simetría y transitividad.*

Demostración.

- *(Simetría)* $t = t' \vdash t' = t$ *aplicando la regla de reemplazamiento a* $a \vdash t = t$*, que se obtiene por reflexividad.*
- *(Transitividad)* $t = t', t' = t'' \vdash t = t''$ *aplicando la regla de reemplazamiento a* $t = t' \vdash t = t'$*, que se obtiene por hipótesis de inducción en la longitud de la cadena de restricciones.*

□

Lema (**Lema de Consistencia).** *Para cualquier sistema de restricciones* $\mathcal{C} = (T, L, \vdash)$ *y cualquier* $u \in \mathcal{P}_{fin}(L)$*, se verifica lo siguiente:*

$$\vdash \exists \overline{x}. \bigwedge u \Rightarrow Con(u)$$

para $\overline{x}$ *definido como las variables locales de* u*. El recíproco es falso en general.*

Demostración. *Vamos a demostrarlo por reducción al absurdo. Supongamos que* $\{x_1, \ldots, x_n\}$ *es el conjunto de las variables libres de* u*, y supongamos que se cumplen:*

(1) $\vdash \exists x_1 \ldots \exists x_n . \bigwedge u$

(2) $\neg Con(u)$

Entonces:

(2) $\Rightarrow u \vdash F$ *(definición de "Con")*
$\Rightarrow \bigwedge u \vdash F$ *(aplicando* $(\wedge \vdash)$ *varias veces)*
$\Rightarrow \exists x_1 \ldots \exists x_n . \bigwedge u \vdash F$ *(aplicando* $(\exists \vdash)$ *n veces)*
$\Rightarrow \vdash F$ ***(Contradicción)*** *(aplicando (1),* ***Corte****)*

Partimos de que u es inconsistente (2)*, se tiene entonces que si* $u \vdash F$ *(por definición del predicado de consistencia) entonces* $\bigwedge u \vdash F$*, aplicando la regla* $(\wedge \vdash)$ *consecutivamente. Aplicamos varias veces la regla* $(\exists \vdash)$ *por cada una de las variables libres de u y se deduce que* $\exists x_1 \ldots \exists x_n . \bigwedge u \vdash F$*. Tenemos que* $\exists x_1 \ldots \exists x_n . \bigwedge u$ *se satisface trivialmente por (1) y* $\exists x_1 \ldots \exists x_n . \bigwedge u \vdash F$*, de lo cual se deduce que* $\vdash F$*. Es falso puesto que* $\nvdash F$ *por definición de F.*

□

A.2 Demostraciones del Capítulo 3

Lema (Lema de Demanda). *Si* $\Pi \square \theta$ *es una respuesta no trivial para un objetivo admisible G de un* $COISS(\mathcal{D})$*-programa y* $X \in DVar_{\mathcal{D}}(G)$ *entonces* $\theta'(X) \neq \bot$ *para toda* $\theta' =_{\setminus evar(G)} \theta$ *dada en la* Definición 49*.*

Demostración. *Sea* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$ *no trivial (es decir,* $Sol_{\mathcal{D}}(\Pi) \neq \emptyset$*). Por la* Definición 49*, existe* $\Pi \square \theta' \in Ans_{\mathcal{P}}(G)$ *tal que* $\theta =_{\setminus evar(G)} \theta'$*. Probamos que* $\theta'(X) \neq \bot$ *razonando mediante inducción en el orden* $\gg_P^+$ *(El cierre transitivo de la relación (ver [50] para más detalles) está bien fundado debido a la propiedad de que no hay ciclos entre variables producidas de objetivos admisibles. Consideramos tres casos para* $X \in DVar_{\mathcal{D}}(G)$ *de acuerdo con la* Definición 48*:*

- $X \in DVar_{\mathcal{D}}(S)$*:*
  Suponemos que $\theta'(X) = \bot$ *y sea* $\mu \in Sol_{\mathcal{D}}(\Pi)$*. Por la* Definición 49*,* $\Pi \models_{\mathcal{D}} S\theta'$*, y entonces* $\theta'\mu \in Sol_{\mathcal{D}}(S)$ *y* $\mu(\theta'(X)) = \mu(\bot) = \bot$*. No obstante, puesto que* $X \in DVar_{\mathcal{D}}(S)$ *y de acuerdo con el item* 1 *en la* Definición 36*, tenemos que* $\mu(\theta'(X)) \neq \bot$*. Por tanto,* $\theta'(X) \neq \bot$*.*

- $(X\overline{a}_k \rightarrow R) \in P$ *con* $k > 0$ *y* $R \in DVar_{\mathcal{D}}(G)$*:*
  A partir de $X \gg_P^+ R$ *y* $R \in DVar_{\mathcal{D}}(G)$*, por* hipótesis de inducción $\theta'(R) \neq \bot$*. Por la* Definición 49*,* $\mathcal{P} \vdash_{\mathcal{D}} \theta'(X)\overline{a_k\theta'} \rightarrow \theta'(R) \Leftarrow \Pi$*. Sin embargo, teniendo en cuenta que* $k > 0$*, solo es posible en* $CRWL(\mathcal{D})$ *si* $\theta'(X) \neq \bot$*.*

- $X = e|_{pos(Y,\tau)}$, $(< e, \underline{case}(\tau, Y, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) > \to R) \in P$ *y* $R \in DVar_{\mathcal{D}}(G)$*: En este caso,* $e = f\overline{e}_n$ *con* $f \in DF^n$*. Como* $X = e|_{pos(Y,\tau)}$*, se tiene que* $X \gg^+_P R$*. Además, como* $R \in DVar_{\mathcal{D}}(G)$*, por* hipótesis de inducción $\theta'(R) \neq \bot$*. Por la* Definición 49 *y la regla* $\mathbf{DF}_{\mathcal{P}}$ *del* $CRWL(\mathcal{D})$*-calculo ilustrado en la Definición 41,* $\mathcal{P} \vdash_{\mathcal{D}} f\ \overline{e_n\theta'} \to \theta'(R) \Leftarrow \Pi$ *usando* $(f\overline{t'}_n \to r \Leftarrow P \,\square\, C) \in [\mathcal{P}]_\bot$ *y se deduce que* $\mathcal{P} \vdash_{\mathcal{D}} e_i\theta' \to t'_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$*. Por otro lado,* $pos(Y, \tau) = i \cdot p$ *con* $1 \leq i \leq n$ *y* $p \in Pos(e_i)$ *porque* $\tau \preceq e$*. Debido a la forma del árbol definicional* $\underline{case}$ *(ver* Definición 46*),* t'_i *tiene un símbolo pasivo* h_j $(1 \leq j \leq k)$ *en la posición* p*. Además, solo puede haber símbolos pasivos* h_j *en* t'_i*. Por lo tanto,* $t'_i \neq \bot$*. Por otra parte, teniendo en cuenta que* $\tau \preceq e$ *con* $X = e_i|_p$*, debe haber los mismos símbolos pasivos y en el mismo orden que anteriormente tenía* $\theta'(X)$ *en la posición* p *de* $e_i\theta'$*. Ésto nos lleva a que* $\mathcal{P} \vdash_{\mathcal{D}} e_i\theta' \to t'_i \Leftarrow \Pi$ *aplica la* $CRWL(\mathcal{D})$*-regla de descomposición* **DC** *en el* $CRWL(\mathcal{D})$*-cálculo ilustrado en la Definición 41 para llegar a* $\mathcal{P} \vdash_{\mathcal{D}} \theta'(X) \to h_j \ldots \Leftarrow \Pi$*. Concluimos que* $\theta'(X) \neq \bot$*.* □

Teorema (**Propiedades de** $COIS$)**.** *Para cualquier* $CFLP(\mathcal{D})$*-programa* $\mathcal{P}$ *dado,* $COIS(\mathcal{P})$ *siempre termina y devuelve otro* $CFLP(\mathcal{D})$*-programa* $\mathcal{P}'$ *tal que* $\mathcal{P}'$ *es un* $COISS(\mathcal{D})$ *y para cualquier c-sentencia* φ*:* $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ *si y solo si* $\mathcal{P}' \vdash_{\mathcal{D}} \varphi$*.*

Demostración. *Por la construcción de* $\mathcal{P}'$ *como un* $COISS(\mathcal{D})$ *usando el* algoritmo de transformación*, debe existir* $n \in \mathbb{N}$ *y* $CFLP(\mathcal{D})$*-programas* $\mathcal{P}_i$ $(0 \leq i \leq n)$ *tales que* $\mathcal{P}_0 = \mathcal{P}$*,* $\mathcal{P}_n = \mathcal{P}'$*, y para todo* $0 < i \leq n$*, la relación entre* $\mathcal{P}_{i-1}$ *y* $\mathcal{P}_i$ *corresponde a la aplicación del tercer caso en la especificación de la función* $COIS$*: existe un patrón de llamada* τ *con raíz* f *y una posición demandada (pero no uniformemente demandada)* $p \in VPos(\tau)$ *por* $(\mathcal{P}_{i-1})_f$ *y* $\mathcal{P}_i = (\mathcal{P}_{i-1} \setminus \{f\}) \cup \mathcal{P}_{f_1} \cup \mathcal{P}_{f_2} \cup \{\tau \to \tau_1, \tau \to \tau_2\}$*, donde:*

- f_1*,* f_2 *son nuevos símbolos de función con la misma aridad que* f*.*
- $\tau_1 = \tau\ \{f \mapsto f_1\}$ *y* $\tau_2 = \tau\ \{f \mapsto f_2\}$*.*
- $\mathcal{P}_{i-1} \setminus \{f\}$ *es el subconjunto de reglas de programa de* $\mathcal{P}_{i-1}$ *que no define* f*.*
- $\mathcal{P}_{f_1}$ *es el subconjunto de reglas de programas de* $\mathcal{P}_{i-1}$ *definiendo* f *que demanda* p*, con todos los lados izquierdos afectados por el cambio* $\{f \mapsto f_1\}$*.*
- $\mathcal{P}_{f_2}$ *es el subconjunto de reglas de programa de* $\mathcal{P}_{i-1}$ *definiendo* f *pero sin demandar* p*, con todos los lados izquierdos afectados por el cambio* $\{f \mapsto f_2\}$*.*

Probamos que para todo $0 < i \leq n$*:* $\mathcal{P}_{i-1} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$ *si y solo si* $\mathcal{P}_i \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$*.*

$\Rightarrow$) *Suponemos que* $\mathcal{P}_{i-1} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$ *(*$0 < i \leq n$*). Razonamos por inducción en el tamaño de esta* $CRWL(\mathcal{D})$*-deducción y consideramos casos acordes con la última regla de inferencia usada en la derivación. Si la* $CRWL(\mathcal{D})$*-regla usada es* **TI**, **RR** *o* **SP** *entonces el resultado se cumple directamente con dicha regla. Si la regla es* **DC**, **IR**, **PF** *o* **AC** *el resultado también se cumple usando la* hipótesis de inducción *y la misma regla de deducción. Finalmente, suponemos que la última regla usada es* $\mathbf{DF}_{\mathcal{P}}$*. Consideramos dos subcasos:*

- Subcaso 1*:* $e = f\overline{e}_n\overline{a}_k$ *con* f *el símbolo de función* $\mathcal{P}_{i-1}$ *afectado por el paso de transformación de* $\mathcal{P}_{i-1}$ *a* $\mathcal{P}_i$*. En ese caso,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}_{\mathbf{i-1}}}$ $(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s])$ *usando* $(f\overline{t}_n \to r \Leftarrow P \;\square\; C) \in [\mathcal{P}_{i-1}]_\perp$*,* $s \in Pat_\perp(\mathcal{U})$ *y donde* $\mathcal{T}_j : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}_c : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} P \;\square\; C \Leftarrow \Pi$*,* $\mathcal{T}_r : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} s\overline{a}_k \to t \Leftarrow \Pi$*. Por* hipótesis de inducción*,* $\mathcal{T}'_j : \mathcal{P}_i \vdash_{\mathcal{D}} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}'_c : \mathcal{P}_i \vdash_{\mathcal{D}} P \;\square\; C \Leftarrow \Pi$*,* $\mathcal{T}'_r : \mathcal{P}_i \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}'_s : \mathcal{P}_i \vdash_{\mathcal{D}} s\overline{a}_k \to t \Leftarrow \Pi$*. Además, desde* $(f\overline{t}_n \to r \Leftarrow P \;\square\; C) \in [\mathcal{P}_{i-1}]_\perp$*, por definición de* $\mathcal{P}_i$*, deben existir instancias* $(f\overline{t}_n \to f_k\overline{t}_n) \in [\mathcal{P}_i]_\perp$ *y* $(f_k\overline{t}_n \to r \Leftarrow P \;\square\; C) \in [\mathcal{P}_i]_\perp$ *con* $k = 1$ *o* 2*. Por ello, podemos construir* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}_\mathbf{i}}$ $(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}'_1, \ldots, \mathcal{T}'_n, \mathcal{T}'_{f_k}, \mathcal{T}'_s])$ *donde* $\mathcal{T}'_{f_k} \equiv \mathbf{DF}_{\mathcal{P}_\mathbf{i}}$ $(f_k\overline{t}_n \to s \Leftarrow \Pi, [\mathcal{T}''_1, \ldots, \mathcal{T}''_n, \mathcal{T}'_c, \mathcal{T}'_r])$ *con* $\mathcal{T}''_j : \mathcal{P}_i \vdash_{\mathcal{D}} t_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*. Por ello,* $\mathcal{T}' : \mathcal{P}_i \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$*.*

- Subcaso 2*:* $e = g\overline{e}_n\overline{a}_k$ *donde* g *es un símbolo de función en* $\mathcal{P}_{i-1}$ *diferente al símbolo de función* f *afectado por el paso de transformación de* $\mathcal{P}_{i-1}$ *a* $\mathcal{P}_i$*. En ese caso,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}_{\mathbf{i-1}}}$ $(g\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s])$ *usando* $(g\overline{t}_n \to r \Leftarrow P \;\square\; C) \in [\mathcal{P}_{i-1}]_\perp$*,* $s \in Pat_\perp(\mathcal{U})$ *y donde* $\mathcal{T}_j : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}_c : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} P \;\square\; C \Leftarrow \Pi$*,* $\mathcal{T}_r : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} s\overline{a}_k \to t \Leftarrow \Pi$*. Por* hipótesis de inducción*,* $\mathcal{T}'_j : \mathcal{P}_i \vdash_{\mathcal{D}} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}'_c : \mathcal{P}_i \vdash_{\mathcal{D}} P \;\square\; C \Leftarrow \Pi$*,* $\mathcal{T}'_r : \mathcal{P}_i \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}'_s : \mathcal{P}_i \vdash_{\mathcal{D}} s\overline{a}_k \to t \Leftarrow \Pi$*. Además, a partir de* $(g\overline{t}_n \to r \Leftarrow P \;\square\; C) \in [\mathcal{P}_{i-1}]_\perp$ *donde* $g \neq f$*, por definición de* $\mathcal{P}_i$*,* $(g\overline{t}_n \to r \Leftarrow P \;\square\; C) \in [\mathcal{P}_i]_\perp$*. Entonces, podemos construir* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}_\mathbf{i}}$ $(g\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}'_1, \ldots, \mathcal{T}'_n, \mathcal{T}'_c, \mathcal{T}'_r, \mathcal{T}'_s])$*. Por ello,* $\mathcal{T}' : \mathcal{P}_i \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$*.*

$\Leftarrow$) *Asumimos que* $\mathcal{P}_i \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$*. Razonamos por inducción en el tamaño de su* $CRWL(\mathcal{D})$*-deducción y distinguimos casos de acuerdo con la última regla de inferencia usada en la derivación. Si la* $CRWL(\mathcal{D})$*-regla usada es* **TI**, **RR** *o* **SP** *entonces el resultado se cumple directamente con la misma regla. En otro caso, si la regla usada es* **DC**, **IR**, **PF** *o* **AC**, *entonces el resultado se cumple aplicando la* hipótesis de inducción *y la misma regla de deducción. Finalmente, suponemos que la última regla usada en la deducción es* $\mathbf{DF}_{\mathcal{P}}$*. Distinguimos dos subcasos diferentes:*

- Subcaso 1*:* $e = f\overline{e}_n\overline{a}_k$ *con* f *el símbolo de función* $\mathcal{P}_{i-1}$ *afectado por el paso de transformación de* $\mathcal{P}_{i-1}$ *a* $\mathcal{P}_i$*. En este caso,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}_\mathbf{i}}\ (f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_{f_k}, \mathcal{T}_s])$ *usando* $(f\overline{t}_n \to f_k\overline{t}_n) \in [\mathcal{P}_i]_\perp$ $(k = 1\ o\ 2)$ *y* $s \in Pat_\perp(\mathcal{U})$*. Entonces,* $\mathcal{T}_j : \mathcal{P}_i \vdash_\mathcal{D} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}_s : \mathcal{P}_i \vdash_\mathcal{D} s\overline{a}_k \to t \Leftarrow \Pi$ *y* $\mathcal{T}_{f_k} \equiv \mathbf{DF}_{\mathcal{P}_\mathbf{i}}\ (f_k\overline{t}_n \to s \Leftarrow \Pi, [\mathcal{T}'_1, \ldots, \mathcal{T}'_n, \mathcal{T}_c, \mathcal{T}_r])$ *con* $\mathcal{T}'_j : \mathcal{P}_i \vdash_\mathcal{D} t_j \to s_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}_c : \mathcal{P}_i \vdash_\mathcal{D} P \ \square\ C \Leftarrow \Pi$ *y* $\mathcal{T}_r : \mathcal{P}_i \vdash_\mathcal{D} r \to s \Leftarrow \Pi$*, usando* $(f_k\overline{s}_n \to r \Leftarrow P \ \square\ C) \in [\mathcal{P}_i]_\perp$*. Por* hipótesis de inducción, $\mathcal{T}^*_j : \mathcal{P}_{i-1} \vdash_\mathcal{D} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$ *and* $\mathcal{T}'^*_j : \mathcal{P}_{i-1} \vdash_\mathcal{D} t_j \to s_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$ *(a partir de aquí, usando las propiedades de* $CRWL(\mathcal{D})$*,* $\mathcal{T}''_j : \mathcal{P}_{i-1} \vdash_\mathcal{D} e_j \to s_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*),* $\mathcal{T}'_c : \mathcal{P}_{i-1} \vdash_\mathcal{D} P \ \square\ C \Leftarrow \Pi$*,* $\mathcal{T}'_r : \mathcal{P}_{i-1} \vdash_\mathcal{D} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}'_s : \mathcal{P}_{i-1} \vdash_\mathcal{D} s\overline{a}_k \to t \Leftarrow \Pi$*. A partir de* $(f\overline{t}_n \to f_k\overline{t}_n) \in [\mathcal{P}_i]_\perp$ *y* $(f_k\overline{s}_n \to r \Leftarrow P \ \square\ C) \in [\mathcal{P}_i]_\perp$ *con* $k = 1\ o\ 2$*, existe una instancia* $(f\overline{s}_n \to r \Leftarrow P \ \square\ C) \in [\mathcal{P}_{i-1}]_\perp$*, debido a la definición de* $\mathcal{P}_i$ *de* $\mathcal{P}_{i-1}$ *y porque* f *es un símbolo de función que aparece en* $\mathcal{P}_{i-1}$ *por hipótesis. Entonces,* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}_\mathbf{i-1}}\ (f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}''_1, \ldots, \mathcal{T}''_n, \mathcal{T}'_c, \mathcal{T}'_r, \mathcal{T}'_s])$*. Pot tanto,* $\mathcal{T}' : \mathcal{P}_{i-1} \vdash_\mathcal{D} e \to t \Leftarrow \Pi$*.*

- Subcaso 2*:* $e = g\overline{e}_n\overline{a}_k$ *donde* g *es un símbolo de función de* $\mathcal{P}_{i-1}$ *diferente del símbolo de función* f *afectado por el paso de transformación de* $\mathcal{P}_{i-1}$ *a* $\mathcal{P}_i$*. En ese caso,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}_\mathbf{i}}\ (g\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s])$ *usando* $(g\overline{t}_n \to r \Leftarrow P \ \square\ C) \in [\mathcal{P}_i]_\perp$*,* $s \in Pat_\perp(\mathcal{U})$ *y donde* $\mathcal{T}_j : \mathcal{P}_i \vdash_\mathcal{D} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}_c : \mathcal{P}_i \vdash_\mathcal{D} P \ \square\ C \Leftarrow \Pi$*,* $\mathcal{T}_r : \mathcal{P}_i \vdash_\mathcal{D} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P}_i \vdash_\mathcal{D} s\overline{a}_k \to t \Leftarrow \Pi$*. Por* hipótesis de inducción, $\mathcal{T}'_j : \mathcal{P}_{i-1} \vdash_\mathcal{D} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}'_c : \mathcal{P}_{i-1} \vdash_\mathcal{D} P \ \square\ C \Leftarrow \Pi$*,* $\mathcal{T}'_r : \mathcal{P}_{i-1} \vdash_\mathcal{D} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}'_s : \mathcal{P}_{i-1} \vdash_\mathcal{D} s\overline{a}_k \to t \Leftarrow \Pi$*. Además, puesto que* $(g\overline{t}_n \to r \Leftarrow P \ \square\ C) \in [\mathcal{P}_i]_\perp$ *donde* $g \neq f$*, por definición de* $\mathcal{P}_i$*,* $(g\overline{t}_n \to r \Leftarrow P \ \square\ C) \in [\mathcal{P}_{i-1}]_\perp$*. Entonces, podemos construir* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}_\mathbf{i-1}}\ (g\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}'_1, \ldots, \mathcal{T}'_n, \mathcal{T}'_c, \mathcal{T}'_r, \mathcal{T}'_s])$*. Por tanto,* $\mathcal{T}' : \mathcal{P}_{i-1} \vdash_\mathcal{D} e \to t \Leftarrow \Pi$*.*

□

Lema 65 (Propiedad de Particionado). *Sea* $\mathcal{P}$ *un* $CFLP(\mathcal{D})$*-programa. Para cualquier* $e \in Exp_\perp(\mathcal{U})$*,* $t \in Pat_\perp(\mathcal{U})$ *y* $p \in Pos(e)$*, las siguientes condiciones son equivalentes:*

1. $\mathcal{T} : \mathcal{P} \vdash_\mathcal{D} e \to t \Leftarrow \Pi$*.*
2. *Existe* $s \in Pat_\perp(\mathcal{U})$ *tal que* $\mathcal{T}_1 : \mathcal{P} \vdash_\mathcal{D} e|_p \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_\mathcal{D} e[s]_p \to t \Leftarrow \Pi$*.*

Además, cuando se demuestra "(1) $\Rightarrow$ *(2)" uno puede escoger* $|\ \mathcal{T}_1\ |, |\ \mathcal{T}_2\ | \leq |\ \mathcal{T}\ |$*.*

Demostración. *Distinguimos varios casos:*

- $p = \epsilon$:
  (1) $\Rightarrow$ (2). *Suponemos que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$. *Tomando* $s \equiv t \in Pat_{\perp}(\mathcal{U})$ *se deduce que* $\mathcal{T}_1 \equiv \mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e|_{\epsilon} = e \to t \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[t]_{\epsilon} = t \to t \Leftarrow \Pi$ *por la* propiedad de aproximación *porque trivialmente* $\Pi \models_{\mathcal{D}} t \sqsupseteq t$. *Observamos que* $\mid \mathcal{T}_1 \mid = \mid \mathcal{T} \mid$ *y* $\mid \mathcal{T}_2 \mid = 0 \leq \mid \mathcal{T} \mid$. *Así pues,* $\mid \mathcal{T}_1 \mid, \mid \mathcal{T}_2 \mid \leq \mid \mathcal{T} \mid$.
  (2) $\Rightarrow$ (1). *Asumimos* $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e|_{\epsilon} = e \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_{\epsilon} = s \to t \Leftarrow \Pi$. *Puesto ques* $\in Pat_{\perp}(\mathcal{U})$, *por la* propiedad de aproximación *tenemos que* $\Pi \models_{\mathcal{D}} s \sqsupseteq t$. *Finalmente, por la* propiedad de encubrimiento *[52], obtenemos* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$.

- $p \neq \epsilon$: *Razonamos por inducción en el tamaño de* $e \in Exp_{\perp}(\mathcal{U})$ *y distinguimos casos dependiendo de la forma de* $t \in Pat_{\perp}(\mathcal{U})$ *y* $\Pi \subseteq PCon_{\perp}(\mathcal{D})$:

- $t = \perp$:
  (1) $\Rightarrow$ (2). *Asumimos* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to \perp \Leftarrow \Pi$. *Tomando* $s \equiv \perp \in Pat_{\perp}(\mathcal{U})$ *se deduce que* $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e|_p \to \perp \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[\perp]_p \to \perp \Leftarrow \Pi$ *usando la regla* **TI**. *En este caso,* $\mid \mathcal{T} \mid = \mid \mathcal{T}_1 \mid = \mid \mathcal{T}_2 \mid = 0$.
  (2) $\Rightarrow$ (1). *Trivialmente,* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to \perp \Leftarrow \Pi$ *usando* **TI**.

- $t \neq \perp$, $InSat_{\mathcal{D}}(\Pi)$:
  (1) $\Leftrightarrow$ (2). *En este caso* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$, $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e|_p \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p \to t \Leftarrow \Pi$ *se deducen trivialmente usando* **TI**. *Además,* $\mid \mathcal{T} \mid = \mid \mathcal{T}_1 \mid = \mid \mathcal{T}_2 \mid = 0$.

- $t \neq \perp$, $Sat_{\mathcal{D}}(\Pi)$. *Distinguimos nuevos casos:*

- $e \in Pat_{\perp}(\mathcal{U})$. *Probamos* (1) $\Rightarrow$ (2):
  En este caso, $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$ *es equivalente a* $\Pi \models_{\mathcal{D}} e \sqsupseteq t$ *por la* propiedad de aproximación. *Puesto que* $p \neq \epsilon$, $e = h\bar{e}_m \in Pat_{\perp}(\mathcal{U})$ *pasivo. Entonces,* $p = i \cdot q$ *con* $1 \leq i \leq m$ *y* $q \in Pos(e_i)$. *Puesto que* $t \neq \perp$ *tenemos dos posibilidades para* t:

- $t = h\bar{t}_m$:
  En este caso $\mathcal{T} \equiv \mathbf{DC}\ (h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$ *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq m$. *Puesto que* $q \in Pos(e_i)$, *por* hipótesis de inducción *existe* $s_i \in Pat_{\perp}(\mathcal{U})$ *tal que* $\mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s_i \Leftarrow \Pi$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s_i]_q \to t_i \Leftarrow \Pi$ *tal que* $\mid \mathcal{T}_{i1} \mid, \mid \mathcal{T}_{i2} \mid \leq \mid \mathcal{T}_i \mid$. *En esta situación,* $e|_p = h\bar{e}_m|_{i \cdot q} = e_i|_q$ *y* $e[s]_p = h\bar{e}_m[s]_{i \cdot q} = h\ e_1 \ldots e_{i-1}\ e_i[s]_q\ e_{i+1} \ldots e_m$. *Así pues, tomando* $s \equiv s_i \in Pat_{\perp}(\mathcal{U})$ *tenemos* $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e|_p = e_i|_q \to s_i = s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p = h\ e_1 \ldots e_{i-1}\ e_i[s_i]_q\ e_{i+1} \ldots e_m \to t \Leftarrow \Pi$ *donde* $\mathcal{T}_2 \equiv \mathbf{DC}\ (e[s_i]_p \to h\bar{t}_m \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$. *Además,* $\mid \mathcal{T}_1 \mid = \mid \mathcal{T}_{i1} \mid \leq \mid \mathcal{T}_i \mid \leq \sum_i \mid \mathcal{T}_i \mid = \mid \mathcal{T} \mid$ *y* $\mid \mathcal{T}_2 \mid = \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_{i2} \mid \leq \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_i \mid = \mid \mathcal{T} \mid$. *Así pues* $\mid \mathcal{T}_1 \mid, \mid \mathcal{T}_2 \mid \leq \mid \mathcal{T} \mid$.

- $t = X \in \mathcal{V}$:
  En este caso $\mathcal{T} \equiv \mathbf{IR}\ (h\bar{e}_m \to X \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$ *donde* $\Pi \models_{\mathcal{D}} h\bar{t}_m \sqsupseteq X$ *y* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq m$. *Puesto que* $q \in Pos(e_i)$, *por* hipótesis de inducción *existe* $s_i \in Pat_\perp(\mathcal{U})$ *tal que* $\mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s_i \Leftarrow \Pi$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s_i]_q \to t_i \Leftarrow \Pi$ *tal que* $\mid \mathcal{T}_{i1} \mid, \mid \mathcal{T}_{i2} \mid \leq \mid \mathcal{T}_i \mid$. *En este caso,* $e|_p = h\bar{e}_m|_{i\cdot q} = e_i|_q$ *y* $e[s]_p = h\bar{e}_m[s]_{i\cdot q} = h\ e_1 \ldots e_{i-1}\ e_i[s]_q\ e_{i+1} \ldots e_m$. *Así pues, tomando* $s \equiv s_i \in Pat_\perp(\mathcal{U})$ *tenemos* $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e|_p = e_i|_q \to s_i = s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p = h\ e_1 \ldots e_{i-1}\ e_i[s_i]_q\ e_{i+1} \ldots e_m \to t \Leftarrow \Pi$ *donde* $\mathcal{T}_2 \equiv \mathbf{IR}\ (e[s_i]_p \to X \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$ *debido a* $\Pi \models_{\mathcal{D}} h\bar{t}_m \sqsupseteq X$. *Además,* $\mid \mathcal{T}_1 \mid = \mid \mathcal{T}_{i1} \mid \leq \mid \mathcal{T}_i \mid \leq \sum_i \mid \mathcal{T}_i \mid = \mid \mathcal{T} \mid$ *y* $\mid \mathcal{T}_2 \mid = \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_{i2} \mid \leq \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_i \mid = \mid \mathcal{T} \mid$. *Así pues* $\mid \mathcal{T}_1 \mid, \mid \mathcal{T}_2 \mid \leq \mid \mathcal{T} \mid$.

- $e \in Pat_\perp(\mathcal{U})$. *Probamos* (2) $\Rightarrow$ (1):
  En este caso $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e|_p = h\bar{e}_m|_{i\cdot q} = e_i|_q \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p = h\bar{e}_m[s]_{i\cdot q} = h\ e_1 \ldots e_{i-1}\ e_i[s]_q\ e_{i+1} \ldots e_m \to t \Leftarrow \Pi$. *Puesto que* $t \neq \perp$ *tenemos de nuevo dos posibilidades para* t:

- $t = h\bar{t}_m$:
  En este caso $\mathcal{T}_2 \equiv \mathbf{DC}\ (h\ e_1 \ldots e_{i-1}\ e_i[s]_q\ e_{i+1} \ldots e_m \to h\bar{t}_m \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$ *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, i-1, i+1, \ldots, m\}$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \to t_i \Leftarrow \Pi$. *Por* hipótesis de inducción *tenemos también* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$. *Entonces,* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi$ *donde* $\mathcal{T} \equiv \mathbf{DC}\ (h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$.

- $t = X \in \mathcal{V}$:
  En este caso $\mathcal{T}_2 \equiv \mathbf{IR}\ (h\ e_1 \ldots e_{i-1}\ e_i[s]_q\ e_{i+1} \ldots e_m \to X \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$ *donde* $\Pi \models_{\mathcal{D}} h\bar{t}_m \sqsupseteq X$, $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, i-1, i+1, \ldots, m\}$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \to t_i \Leftarrow \Pi$. *Por* hipótesis de inducción *tenemos también* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$. *Entonces,* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} h\bar{e}_m \to X \Leftarrow \Pi$ *donde* $\Pi \models_{\mathcal{D}} h\bar{t}_m \sqsupseteq X$ *y* $\mathcal{T} \equiv \mathbf{IR}\ (h\bar{e}_m \to X \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$.

- $e \notin Pat_\perp(\mathcal{U})$. *Distinguimos nuevos casos:*

- $e = h\bar{e}_m \notin Pat_\perp(\mathcal{U})$ *pasivo. Este caso es análogo al caso anterior (Puesto que* $t \neq \perp$, $t = h\bar{t}_m$ *o* $t = X \in \mathcal{V}$*).*

- $e = f\bar{e}_n\bar{a}_k$ *con* $f \in DF^n$ $(k \geq 0)$. *Demostramos* (1) $\Rightarrow$ (2):
  Asumimos $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} f\bar{e}_n\bar{a}_k \to t \Leftarrow \Pi$. *En este caso* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ (f\bar{e}_n\bar{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s\ ])$ *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$, $\mathcal{T}_p : \mathcal{P} \vdash_{\mathcal{D}} P\ \square\ C \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} s\bar{a}_k \to t \Leftarrow \Pi$, *usando* $(f\bar{t}_n \to r \Leftarrow P\ \square\ C) \in [\mathcal{P}]_\perp$ *y* $s \in Pat_\perp(\mathcal{U})$. *Puesto que* $p = i \cdot q$ *con* $i \in \{1, \ldots, n+k\ \}$, *distinguimos dos casos:*

- $1 \leq i \leq n$:
  Puesto que $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *y* $q \in Pos(e_i)$, *por* hipótesis de inducción *se tiene que existe* $s_i \in Pat_\perp(\mathcal{U})$ *tal que* $\mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s_i \Leftarrow \Pi$, $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s_i]_q \to t_i \Leftarrow \Pi$ *y* $\mid \mathcal{T}_{i1} \mid, \mid \mathcal{T}_{i2} \mid \leq \mid \mathcal{T}_i \mid$. *En esta situación* $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k|_{i\cdot q} = e_i|_q \to s_i \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k[s_i]_p \to t \Leftarrow \Pi$ *donde* $\mathcal{T}_2 \equiv \mathbf{DF}_{\mathcal{P}}$ $(fe_1 \ldots e_{i-1}\ e_i[s_i]_q\ e_{i+1} \ldots e_n\ \overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s\ ])$, *usando* $(f\overline{t}_n \to r \Leftarrow P \ \square\ C) \in [\mathcal{P}]_\perp$ *y* $s \in Pat_\perp(\mathcal{U})$. *Además,* $\mid \mathcal{T}_1 \mid = \mid \mathcal{T}_{i1} \mid \leq \mid \mathcal{T}_i \mid \leq \mid \mathcal{T}_i \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_c \mid + \mid \mathcal{T}_r \mid + \mid \mathcal{T}_s \mid + 1 = \mid \mathcal{T} \mid$ *y* $\mid \mathcal{T}_2 \mid = 1 + \mid \mathcal{T}_{i2} \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_c \mid + \mid \mathcal{T}_r \mid + \mid \mathcal{T}_s \mid \leq 1 + \mid \mathcal{T}_i \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_c \mid + \mid \mathcal{T}_r \mid + \mid \mathcal{T}_s \mid = \mid \mathcal{T} \mid$. *Así pues,* $\mid \mathcal{T}_1 \mid, \mid \mathcal{T}_2 \mid \leq \mid \mathcal{T} \mid$.

- $n+1 \leq i \leq k+1$:
  Sabemos que $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} s\overline{a}_k \to t \Leftarrow \Pi$. *Puesto que* $q \in Pos(a_i)$, *existe* $q' \in Pos(s\overline{a}_k)$. *Por* hipótesis de inducción *existe* $s_i \in Pat_\perp(\mathcal{U})$ *tal que* $\mathcal{T}_{s1} : \mathcal{P} \vdash_{\mathcal{D}} s\overline{a}_k|_{q'} = a_i|_q \to s_i \Leftarrow \Pi$, $\mathcal{T}_{s2} : \mathcal{P} \vdash_{\mathcal{D}} s\overline{a}_k[s_i]_{q'} = s\ a_1 \ldots a_{i-1}\ a_i[s_i]_q\ a_{i+1} \ldots a_k \to t \Leftarrow \Pi$ *y* $\mid \mathcal{T}_{s1} \mid, \mid \mathcal{T}_{s2} \mid \leq \mid \mathcal{T}_s \mid$. *En esta situación* $\mathcal{T}_1 \equiv \mathcal{T}_{s1} : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k|_{i\cdot q} = a_i|_q \to s_i \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k[s_i]_{i\cdot q} \to t \Leftarrow \Pi$ *donde* $\mathcal{T}_2 \equiv \mathbf{DF}_{\mathcal{P}}$ $(f\overline{e}_n\ a_1 \ldots a_{i-1}\ a_i[s_i]_q\ a_{i+1} \ldots a_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s2}])$. *Además,* $\mid \mathcal{T}_1 \mid = \mid \mathcal{T}_{s1} \mid \leq \mid \mathcal{T}_s \mid \leq 1 + \sum_i \mid \mathcal{T}_i \mid + \mid \mathcal{T}_c \mid + \mid \mathcal{T}_r \mid + \mid \mathcal{T}_s \mid = \mid \mathcal{T} \mid$ *y* $\mid \mathcal{T}_2 \mid = 1 + \sum_i \mid \mathcal{T}_i \mid + \mid \mathcal{T}_c \mid + \mid \mathcal{T}_r \mid + \mid \mathcal{T}_{s2} \mid \leq 1 + \sum_i \mid \mathcal{T}_i \mid + \mid \mathcal{T}_c \mid + \mid \mathcal{T}_r \mid + \mid \mathcal{T}_s \mid = \mid \mathcal{T} \mid$. *Así pues,* $\mid \mathcal{T}_1 \mid, \mid \mathcal{T}_2 \mid \leq \mid \mathcal{T} \mid$.

- $e = f\overline{e}_n\overline{a}_k$ *con* $f \in DF^n$ $(k \geq 0)$. *Demostramos* $(2) \Rightarrow (1)$:
  Asumimos $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k|_p \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k[s]_p \to t \Leftarrow \Pi$. *Puesto que* $p = i \cdot q$ *con* $i \in \{1, \ldots, n+k\}$, *de nuevo distinguimos dos casos:*

- $1 \leq i \leq n$:
  En este caso $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\ e_1 \ldots e_{i-1}\ e_i[s]_q\ e_{i+1} \ldots e_n\ \overline{a}_k \to t \Leftarrow \Pi$. *Observamos que* $\mathcal{T}_2 \equiv \mathbf{DF}_{\mathcal{P}}$ $(f\ e_1 \ldots e_{i-1}\ e_i[s]_q\ e_{i+1} \ldots e_n\ \overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s'}])$ *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, i-1, i+1, \ldots, n\}$, $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \to t_i \Leftarrow \Pi$, $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P \ \square\ C \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s' \Leftarrow \Pi$ *y* $\mathcal{T}_{s'} : \mathcal{P} \vdash_{\mathcal{D}} s'\overline{a}_k \to t \Leftarrow \Pi$ *usando* $(f\overline{t}_n \to r \Leftarrow P \ \square\ C) \in [\mathcal{P}]_\perp$ *y* $s' \in Pat_\perp(\mathcal{U})$. *Puesto que* $\mathcal{T}_{i1} \equiv \mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s \Leftarrow \Pi$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \to t_i \Leftarrow \Pi$, *por* hipótesis de inducción $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$. *Entonces,* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$ *donde* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}$ $(f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s'}])$.

- $n+1 \leq i\ k+1$:
  En este caso $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} a_i|_q \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\ \overline{e}_n\ a_1 \ldots a_{i-1}\ a_i[s]_q\ a_{i+1} \ldots a_k \to t \Leftarrow \Pi$. *Observamos que* $\mathcal{T}_2 \equiv \mathbf{DF}_{\mathcal{P}}$ $(f\ \overline{e}_n\ a_1 \ldots a_{i-1}\ a_i[s]_q\ a_{i+1} \ldots a_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s'2}])$ *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, n\}$, $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P \ \square\ C \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s' \Leftarrow \Pi$ *y* $\mathcal{T}_{s'2} : \mathcal{P} \vdash_{\mathcal{D}} s'\ a_1 \ldots a_{i-1}\ a_i[s]_q\ a_{i+1} \ldots a_k \to t \Leftarrow \Pi$ *usando* $(f\overline{t}_n \to r \Leftarrow P \ \square\ C)$

$\in [\mathcal{P}]_\perp$ *y* $s' \in Pat_\perp(\mathcal{U})$*. Puesto que* $q \in Pos(a_i)$*, existe* $q' \in Pos(s'\overline{a}_k)$ *tal que* $\mathcal{T}_{s'1} \equiv \mathcal{T}_1 : \mathcal{P} \vdash_\mathcal{D} s'\overline{a}_k|_{q'} = a_i|_q \to s \Leftarrow \Pi$ *y* $\mathcal{T}_{s'2} : \mathcal{P} \vdash_\mathcal{D} s'\overline{a}_k[s]_{q'} = s'\ a_1\ \ldots\ a_{i-1}\ a_i[s]_q\ a_{i+1}\ \ldots\ a_k \to t \Leftarrow \Pi$*. Por* hipótesis de inducción $\mathcal{T}_{s'} : \mathcal{P} \vdash_\mathcal{D} s'\overline{a}_k \to t \Leftarrow \Pi$*. Entonces,* $\mathcal{T} : \mathcal{P} \vdash_\mathcal{D} f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$ *donde* $\mathcal{T} \equiv \mathbf{DF}_\mathcal{P}\ (f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s'}])$.

- $e = p\overline{e}_n$ *con* $p \in PF^n$*. Demostramos* (1) $\Rightarrow$ (2)*:*
  Asumimos $\mathcal{T} : \mathcal{P} \vdash_\mathcal{D} p\overline{e}_n \to t \Leftarrow \Pi$*. En este caso* $\mathcal{T} \equiv \mathbf{PF}\ (p\overline{e}_n \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$ *donde* $\mathcal{T}_i : \mathcal{P} \vdash_\mathcal{D} e_i \to t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$ *y* $\Pi \models_\mathcal{D} p\overline{t}_n \to t$*. Puesto que* $i \cdot q \in Pos(p\overline{e}_n)$*, tenemos* $q \in Pos(e_i)$*. Por* hipótesis de inducción*, existe* $s_i \in Pat_\perp(\mathcal{U})$ *tal que* $\mathcal{T}_{i1} : \mathcal{P} \vdash_\mathcal{D} e_i|_q \to s_i \Leftarrow \Pi$*,* $\mathcal{T}_{i2} : \mathcal{P} \vdash_\mathcal{D} e_i[s_i]_q \to t_i \Leftarrow \Pi$ *y* $\mid \mathcal{T}_{i1} \mid, \mid \mathcal{T}_{i2} \mid \leq \mid \mathcal{T}_i \mid$*. Tomando* $s \equiv s_i$*, tenemos* $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P} \vdash_\mathcal{D} p\overline{e}_n|_{i\cdot q} = e_i|_q \to s_i \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_\mathcal{D} p\overline{e}_n[s_i]_{i\cdot q} \to t \Leftarrow \Pi$ *donde* $\mathcal{T}_2 \equiv \mathbf{PF}\ (p\ e_1\ \ldots\ e_{i-1}\ e_i[s_i]_q\ e_{i+1}\ \ldots\ e_n \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n\ ])$ *puesto que* $\Pi \models_\mathcal{D} p\overline{t}_n \to t$*. Además,* $\mid \mathcal{T}_1 \mid = \mid \mathcal{T}_{i1} \mid \leq \mid \mathcal{T}_i \mid \leq \mid \mathcal{T}_i \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid + 1 = \mid \mathcal{T} \mid y \mid \mathcal{T}_2 \mid = 1 + \mid \mathcal{T}_{i2} \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid \leq 1 + \mid \mathcal{T}_i \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid = \mid \mathcal{T} \mid$*. Así pues,* $\mid \mathcal{T}_1 \mid, \mid \mathcal{T}_2 \mid \leq \mid \mathcal{T} \mid$.

- $e = p\overline{e}_n$ *con* $p \in PF^n$*. Demostramos* (2) $\Rightarrow$ (1)*:*
  Asumimos $\mathcal{T}_1 : \mathcal{P} \vdash_\mathcal{D} p\overline{e}_n|_{i\cdot q} = e_i|_q \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_\mathcal{D} p\overline{e}_n[s]_{i\cdot q} = p\ e_1\ \ldots\ e_{i-1}\ e_i[s]_q\ e_{i+1}\ \ldots\ e_n \to t \Leftarrow \Pi$ *con* $s \in Pat_\mathcal{D}(\mathcal{U})$ *y* $q \in Pos(e_i)$*. En esta situación,* $\mathcal{T}_2 \equiv \mathbf{PF}\ (p\ e_1\ \ldots\ e_{i-1}\ e_i[s]_q\ e_{i+1}\ \ldots\ e_n \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n\ ])$ *donde* $\mathcal{T}_i : \mathcal{P} \vdash_\mathcal{D} e_i \to t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, i-1, i+1, \ldots, n\}$*,* $\mathcal{T}_{i2} : \mathcal{P} \vdash_\mathcal{D} e_i[s]_q \to t_i \Leftarrow \Pi$ *y* $\Pi \models_\mathcal{D} p\overline{t}_n \to t$*. Puesto que* $\mathcal{T}_{i1} \equiv \mathcal{T}_1 : \mathcal{P} \vdash_\mathcal{D} e_i|_q \to s \Leftarrow \Pi$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_\mathcal{D} e_i[s]_q \to t_i \Leftarrow \Pi$*, por* hipótesis de inducción*,* $\mathcal{T}_i : \mathcal{P} \vdash_\mathcal{D} e_i \to t_i \Leftarrow \Pi$*. Entonces,* $\mathcal{T} : \mathcal{P} \vdash_\mathcal{D} p\overline{e}_n \to t \Leftarrow \Pi$ *tomando* $\mathcal{T} \equiv \mathbf{PF}\ (p\overline{e}_n \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n\ ])$ *porque* $\Pi \models_\mathcal{D} p\overline{t}_n \to t$.

El resto de este apéndice presenta los principales resultados de este trabajo, que son la *corrección* y la *completitud* de la resolución de objetivos en $CDNC(\mathcal{D})$ con respecto a $CRWL(\mathcal{D})$-semánticas. Para demostrar ambas propiedades usamos técnicas similares a las usadas por el $CLNC(\mathcal{D})$-cálculo presentado en [51]. El primer resultado demuestra la corrección para un único paso de transformación. Esto quiere decir que los pasos de transformación preservan la admisibilidad de los objetivos, fallan solo en el caso de que los objetivos sean insatisfactibles y no introducen nuevas soluciones.

Lema 66 (Lema de Corrección).

1. *Los pasos de transformación preservan la admisibilidad de los objetivos: Si* $G \vdash\!\!\!\vdash_{CDNC(\mathcal{D})} G'$ *y* G *es un objetivo admisible, entonces* G' *es un objetivo admisible.*

2. *Los pasos de transformación fallan solo en el caso de los objetivos insatisfactibles: Si* $G \Vdash_{CDNC(\mathcal{D})} \blacksquare$ *entonces* $Sol_{\mathcal{P}}(G) = \emptyset$ *(o equivalentemente,* $Ans_{\mathcal{P}}(G)$ *incluye solo respuestas triviales).*

3. *Los pasos de transformación no añaden nuevas respuestas: Si* $G \Vdash_{CDNC(\mathcal{D})} G'$ *y* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$ *entonces* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$.

Demostración. *Para esta demostración supondremos, para cada* $CDNC(\mathcal{D})$*-regla, que* G, G' *y* G_i *son exáctamente como aparecen en la presentación del* $CDNC(\mathcal{D})$*-cálculo.*

(1) Para cada $CDNC(\mathcal{D})$*-regla damos explicaciones concisas justificando la preservación de las condiciones de admisibilidad dadas en la* Subsección 3.4.1.

SS
LN: *Puesto que* X *es producida y debido a la linealidad de* G, σ_0 *no modifica los lados derechos de* P.
EX: $pvar(P') = pvar(P) \setminus \{X\} \subseteq evar(G) \setminus \{X\} = evar(G')$.
NC: *Puesto que* X *es producida, sólo producciones* $e \to Z$ *con* $X \in var(e)$ *son afectadas por* σ_0. *En estos casos, para cada* $Y \in var(t)$, *la relación* $Y \gg_P Z$ *es creada, donde previamente teníamos* $Y \gg_P X$, $X \gg_P Z$. *Por tanto,* $\gg_P^*$ *no se ve alargada.*
SL: σ_0 *no modifica* σ *y* $pvar(P') \subseteq pvar(P)$.
DT: σ_0 *no modifica las variables de los árboles de producciones demandadas en* P. *Mas aún,* $t \to X$ *no es una producción demandada y* X *no ocurre en* $pvar(P)$ *debido a la linealidad de* G.

IM
LN: *Debido a la linealidad de* G *y* $X_1, \ldots, X_m$ *nuevas variables,* σ_0 *no modifica las variables en los lados derechos de las producciones en* P *y cada nueva variable es producida sólo una vez.*
EX: $pvar(P') = (pvar(P) \setminus \{X\}) \cup \{\overline{X}_m\} \subseteq (evar(G) \setminus \{X\}) \cup \{\overline{X}_m\} = evar(G')$.
NC: *Puesto que* X *es producida,* X *no ocurre en* $\overline{e}_m$. *Así, sólo producciones* $e \to X'$ *en* P *con* $X \in var(e)$ *son afectadas por* σ_0. *En estos casos, para cada* $Y \in var(h\overline{e}_m)$, *la relación* $Y \gg_P X'$ *es creada, donde previamente teníamos* $Y \gg_P X$ *y* $X \gg_P X'$. *Por tanto,* $\gg_P^*$ *no se ve alargada.*
SL: σ_0 *no modifica* σ *y* $pvar(P') = (pvar(P) \setminus \{X\}) \cup \{\overline{X}_m\}$ *con* $\overline{X}_m$ *variables nuevas.*
DT: σ_0 *sólo introduce variables nuevas y no modifica las variables en los árboles de producciones demandadas en* P. *Mas aún, todas las producciones introducidas no tienen un árbol. Del mismo modo,* $h\overline{e}_m \to X$ *no es una producción con un árbol y* X *no ocurre en* $pvar(P)$ *debido a la linealidad de* G.

EL
LN, NC, SL, DT: *Trivial.*
EX: $pvar(P') = pvar(P) \setminus \{X\} \subseteq evar(G) \setminus \{X\} = evar(G')$.

PF
LN: *Trivial, pues* $\overline{X}_q$ *son variables nuevas y distintas.*
EX: *Todas las variables nuevas* $\overline{X}_q$ *están cuantificadas existencialmente.*
NC: *Las variables* $\overline{X}_q$ *son nuevas, y por tanto no aparecen en ningún lado izquierdo de producciones en* G'. *Por tanto, ningún ciclo entre variables producidas se crea.*
SL: σ *no se modifica y las variables* $\overline{X}_q$ *son nuevas.*
DT: *Trivial, pues* $p\overline{e}_n \to X$ *no es una producción con un árbol, todas las producciones introducidas* $e_q \to X_q$ *no tienen un árbol y las* $\overline{X}_q$ *son variables nuevas.*

DT_1
LN, EX, NC, SL: *Trivial.*
DT: *Todas las variables en* $\mathcal{T}_{f\overline{X}_n}$ *son nuevas y* $X \in DVar_{\mathcal{D}}(P \square S)$.

DT_2
LN: *Trivial, pues* X' *es una variable nueva.*
EX: *La variable nueva* X' *está cuantificada existencialmente.*
NC: *Puesto que* X' *es una variable producida nueva, sólo puede aparecer en* $X'\overline{a}_k \to X$. *En este caso, para cada variable* $Y \in var(f\overline{e}_n)$, *la relación* $Y \gg_P X'$, $X' \gg_P X$ *es creada, donde previamente teníamos* $Y \gg_P X$. *Por tanto,* $\gg_P^*$ *no presenta ciclos pues no se ve alargada.*
SL: σ *no se modifica y la variable* X' *es nueva.*
DT: *La variable* X' *es demandada pues existe una suspensión* $X'\overline{a}_k \to X$ *con* $X \in DVar_{\mathcal{D}}(P \square S)$ *(ver* Definición 48*) y todas las variables en* $\mathcal{T}_{f\overline{X}_n}$ *son nuevas.*

FV
LN: *Puesto que* F *no es producida,* σ_0 *no modifica los lados derechos de las producciones en* G'.
EX: *Por la misma razón,* $pvar(P') = \{X\} \cup pvar(P) \subseteq evar(G) \subseteq evar(G) \cup \{\overline{X}_p\} = evar(G')$.
NC: *Puesto que* F *no es producida, solo producciones* $e \to Z \in P$ *con* $F \in var(e)$ *son afectadas por* σ_0. *En estos casos, para cada* $Y \in var(h\overline{X}_p)$, *la relación* $Y \gg_P Z$ *es creada. Pero* $Y \gg_P Z$ *no puede tomar parte de un ciclo de variables, pues cada variable* $Y \in var(h\overline{X}_p)$ *es nueva y no aparece en ningún lado derecho de una producción en el nuevo objetivo.*
SL: *Puesto que* F *no es producida y* $\overline{X}_p$ *son variables nuevas,* σ_0 *puede sólo introducir variables no producidas y nuevas en la sustitución respuesta.*
DT: *Por la misma razón,* σ_0 *sólo introduce variables nuevas y no modifica las variables en los árboles de producciones demandadas en* P.

CSS
LN, EX, NC, SL: *Trivial.*
DT: *Trivial. P y S no se modifican y puesto que $e|_{pos(X,\tau)}$ no es una variable, el árbol $\underline{case}$ $(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$ no define una variable demandada.*

DI
LN: *Puesto que Y no es producida, σ_0 no modifica las variables en los lados derechos de las producciones de P.*
EX: *Por la misma razón, $pvar(P)$ no se modifica, mientras que $R, \overline{U}$ es alargada a $\overline{Y}_{m_i}, R, \overline{U}$.*
NC: *Puesto que G es un objetivo admisible y $\overline{Y}_{m_i}$ son variables nuevas, después de la aplicación de σ_0, estas variables pueden aparecer sólo en los lados izquierdos de las producciones en P' pero no en los lados derechos. Por tanto, ningún ciclo de variables producidas es creado en G'.*
SL: *Ni Y ni los $\overline{Y}_{m_i}$ son variables producidas.*
DT: *σ_0 sólo introduce variables nuevas y el árbol $\underline{case}$ $(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$ define la variable $Y \notin pvar(P)$.*

DN
LN: *R' es una variable nueva.*
EX: *$pvar(P') = pvar(P) \cup \{R\} \cup \{R'\} \subseteq evar(G) \cup \{R'\} = evar(G')$.*
NC: *Puesto que $R' \in var(e[R']_{pos(X,\tau)})$, se tiene que $R' \gg_P R$. En este caso, para cada $Y \in var(e|_{pos(X,\tau)})$, se tiene que $Y \gg_P R'$. Sin embargo, como $Y \in var(e)$, se sigue que $Y \gg_P R$ directamente. Del mismo modo, R no aparece en e (y de este modo no aparece en $e|_{pos(X,\tau)}$) debido a la condición de admisibilidad* **NC**, *y R' no aparece en e ya que es una variable nueva. Por tanto, ningún ciclo de variables producidas es creado.*
SL: *σ no se modifica y la variable R' es nueva.*
DT: *R' es una variable nueva. Mas aún, R' es una variable demandada pues $R' = e[R']_{pos(X,\tau)}|_{pos(X,\tau)}$ en $< e[R']_{pos(X,\tau)}, \underline{case}\ (\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) > \rightarrow R$ siendo R demandada. Finalmente, ni P ni S son modificadas y $e|_{pos(X,\tau)}$ no es una variable.*

RRA
LN: *Puesto que $\tau \rightarrow r_i \Leftarrow P_i \ \square \ C_i$ es una variante de una regla de programa en $\mathcal{P}$, τ es lineal y por tanto $\overline{R}_m \in var(\tau)$ son variables nuevas y distintas. Además, las variables en el lado derecho de P_i son todas también nuevas y distintas de las variables de τ (ver item 2.(a) de la definición de $CFLP(\mathcal{D})$-programa) y por tanto de las variables $\overline{R}_m$. Por el mismo motivo, σ_c no modifica las variables en los lados derechos de P_i.*
EX: *$pvar(P') = pvar(P) \cup pvar(P_i\sigma_c) \cup \{\overline{R}_m\} \cup \{R\} = (pvar(P) \cup \{R\}) \cup (pvar(P_i) \cup \{\overline{R}_m\}) \subseteq evar(G) \cup \{\overline{X}\} = evar(G')$.*

NC: *Ni R ni las variables nuevas $\overline{R}_m$ ocurren en P y e (y de este modo no ocurren en $\sigma_f(R_j)$) debido a la condición de admisibilidad* **NC** *en G. Si $R_j \in var(\sigma_c(r_i))$, $R_j \gg_P R$ y para cada $Y \in var(\sigma_f(R_j))$, $Y \gg_P R_j$. Sin embargo, como $Y \in var(e)$, se sigue que $Y \gg_P R$ directamente. Por otro lado, R no aparece en $P_i\sigma_c$ y los lados derechos de $P_i\sigma_c$ no aparecen en $var(\sigma_f(R_j))$. Por tanto, $\gg_P^*$ no se ve alargada.*
SL: *σ no se ve modificada y tanto $\overline{R}_m$ como las variables producidas de $P_i\sigma_c$ son nuevas.*
DT: *Las variables en rule $(\tau \rightarrow r_1 \Leftarrow P_1 \ \square \ C_1 \mid \ldots \mid r_k \Leftarrow P_k \ \square \ C_k)$ no aparecen en el resto del objetivo. Cuando ésta es eliminada, estas variables son nuevas en el objetivo G'. Además, en $P_i\sigma_c$ no hay producciones con árbol asociado.*

CS
LN: *Puesto que $\chi = pvar(P)$ y el resolutor de restricciones satisface el requerimiento $\chi \cap (Dom(\sigma_i) \cup ran(\sigma_i)) = \emptyset$ dado en el item 2.(b) de la* Definición 36, *tenemos que $pvar(P) \cap Dom(\sigma_i) = \emptyset$ y entonces σ_i no modifica los lados derechos de las producciones en P.*
EX: *Por la misma razón, $pvar(P_i) = pvar(P) \subseteq evar(G) \subseteq evar(G) \cup \{\overline{Y}_i\} = evar(G_i)$.*
NC: *Puesto que también $pvar(P) \cap ran(\sigma_i) = \emptyset$, y $\overline{Y}_i$ son nuevas, ninguna variable producida es introducida por σ_i en los nuevos lados izquierdos de producciones en P. De aquí, ningún ciclo entre variables producidas es creado en G.*
SL: *Puesto que $pvar(P_i) = pvar(P)$ y $pvar(P) \cap var(\sigma_i) = \emptyset$, ninguna variable producida entra en la nueva sustitución respuesta.*
DT: *σ_i sólo introduce variables nuevas y no modifica los lados derechos de las producciones en P. Además, de acuerdo con el item 2.(a) de la* Definition 36, *tenemos que $var(S_i) \cap pvar(P) = \emptyset$ o bien $DVar_{\mathcal{D}}(S_i) \cap pvar(P) \neq \emptyset$. Puesto que los lados derechos de las producciones con árbol son variables producidas y demandadas, si son demandadas por S entonces seguirán apareciendo en S_i y por tanto, seguirán siendo demandadas en G_i por S_i. Analogamente se razonaría si son demandadas por P.*

AC
LN: *Trivial, pues $\overline{X}_q$ son variables nuevas y distintas.*
EX: *Todas las variables nuevas $\overline{X}_q$ están cuantificadas existencialmente.*
NC: *Las variables $\overline{X}_q$ son nuevas, y por tanto no aparecen en ningún lado izquierdo de producciones en G'. Por tanto, ningún ciclo entre variables producidas se crea.*
SL: *σ no se modifica y las variables $\overline{X}_q$ son nuevas.*
DT: *Trivial, pues todas las producciones nuevas introducidas no tienen un árbol, las $\overline{X}_q$ son variables nuevas, y ni P ni S son modificadas.*

(2) Continuamos considerando $CDNC(\mathcal{D})$-reglas de fallo una a una.

CC *Asumamos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$. *Por definición, existe una sustitución* $\hat{\theta}$ *tal que* $\hat{\theta} =_{\setminus evar(G)} \theta$ *y* $< e, \underline{case}\ (\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) > \to R$ *is* $CRWL(\mathcal{D})$*-probable para* $e\hat{\theta} \to R\hat{\theta} \Leftarrow \Pi$, *usando en el último paso la* $CRWL(\mathcal{D})$*-regla* $\mathbf{DF}_{\mathcal{P}}$ *con una instancia* $(f\bar{t}_n \to r \Leftarrow P \ \square \ C) \in [\mathcal{P}]_{\perp}$ *(*$e\hat{\theta} = f\overline{e_n\hat{\theta}}$ *y* R *es una variable demandada con* $\hat{\theta}(R) \neq \perp$ *por el* Lema de Demanda*). Se sigue que* $\mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to t_i \Leftarrow \Pi$ *es* $CRWL(\mathcal{D})$*-probable para todo* $1 \leq i \leq n$. *Como* $e|_{pos(X,\tau)} = h \ldots$, $e\hat{\theta}$ *tiene el símbolo* h *en un argumento* $e_i\hat{\theta}$. *Como* $\tau \preccurlyeq f\bar{t}_n$, $f\bar{t}_n$ *tiene el símbolo* h_j $(1 \leq j \leq k)$ *en el argumento* t_i. *Como* $h \notin \{h_1, \ldots, h_k\}$, *no puede ser cierto que* $\mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to t_i \Leftarrow \Pi$ *sea* $CRWL(\mathcal{D})$*-probable si* $Sat_{\mathcal{D}}(\Pi)$. *Por tanto,* $Ans_{\mathcal{P}}(G)$ *incluye sólo respuestas triviales.*

SF *Asumamos* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$. *Por definición, existe una sustitución* $\hat{\theta}$ *tal que* $\hat{\theta} =_{\setminus evar(G)} \theta$ *y* $\Pi \models_{\mathcal{D}} S\hat{\theta}$. *Sea* $\mu \in Sol_{\mathcal{D}}(\Pi)$. *Entonces* $\hat{\theta}\mu \in Sol_{\mathcal{D}}(S)$. *Puesto que* $Solve^{\mathcal{D}}(S, \chi) = \blacklozenge$, *tenemos que* $Sol_{\mathcal{D}}(S) = \emptyset$ *(debido a los requerimientos del resolutor). De aquí, también* $Sol_{\mathcal{D}}(S\hat{\theta}) = \emptyset$ *y entonces* $Sol_{\mathcal{D}}(\Pi) = \emptyset$ *(ésto implica* $InSat_{\mathcal{D}}(\Pi)$*). Por tanto,* $Ans_{\mathcal{P}}(G)$ *incluye sólo respuestas triviales.*

En ambos casos, $Ans_{\mathcal{P}}(G)$ *sólo incluye respuestas triviales, es decir,* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$ *si* $InSat_{\mathcal{D}}(\Pi)$. *Veamos que* $Sol_{\mathcal{P}}(G) = \emptyset$*: sabemos que* $\mu \in Sol_{\mathcal{P}}(G) \Leftrightarrow (\emptyset \ \square \ \mu) \in Ans_{\mathcal{P}}(G)$. *Obviamente,* $Sol_{\mathcal{D}}(\emptyset) = Val_{\perp}(\mathcal{D})$ *y se tiene que* $Sat_{\mathcal{D}}(\emptyset)$. *Luego* $(\emptyset \ \square \ \mu) \notin Ans_{\mathcal{P}}(G)$ *pues no es una respuesta trivial y en consecuencia* $\mu \notin Sol_{\mathcal{P}}(G)$. *Así pues,* $Sol_{\mathcal{P}}(G) = \emptyset$.

(3) De nuevo continuamos regla a regla.

SS *Asumamos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G')$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\sigma_0\hat{\theta} \Leftarrow \Pi$. *Definimos* $\hat{\theta}'(X) = t\hat{\theta}$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para* $Y \neq X$. *Se cumple entonces que* $\sigma_0\hat{\theta} = \hat{\theta}'$. *Entonces,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta}' \Leftarrow \Pi$. *Puesto que* $X \in pvar(P)$, *se tiene que* $X \notin var(\sigma)$ *por la condición de admisibilidad* **SL**, *por lo que* $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $Y \mapsto s \in \sigma$. *Además, como* $X \notin var(t)$, $X\hat{\theta}' \equiv t\theta \equiv t\hat{\theta}'$. *Entonces,* $\Pi \models_{\mathcal{D}} t\hat{\theta}' \sqsupseteq X\hat{\theta}'$, *y usando la* Propiedad de Aproximación, *también se tiene que* $\mathcal{P} \vdash_{\mathcal{D}} (t \to X)\hat{\theta}' \Leftarrow \Pi$. *Finalmente, como* $\hat{\theta} =_{\setminus \{X\}} \hat{\theta}'$ *concluimos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$.

IM *Asumamos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G')$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\sigma_0\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T}_i \equiv \mathcal{P} \vdash_{\mathcal{D}} (e_i \to X_i)\sigma_0\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq m$. *Puesto que* $X \notin var(e_i)$ *por la condición de admisibilidad* **NC** *y* $\overline{X}_m$ *son variables nuevas en* G', *tenemos también que* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \to X_i)\hat{\theta} \Leftarrow \Pi$. *Ahora, definimos* $\hat{\theta}'(X) = h\overline{X_m\hat{\theta}}$, $\hat{\theta}'(X_i) = X_i$ *para todo* $1 \leq i \leq m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para toda* $Y \notin \{X, \overline{X}_m\}$. *Se cumple que* $\sigma_0\hat{\theta} =_{\setminus \{\overline{X}_m\}}$

$\hat{\theta}'$. *Como* $\overline{X}_m$ *son variables nuevas en* G', $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta}' \Leftarrow \Pi$ *y* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \to X_i\hat{\theta} \Leftarrow \Pi$. *Mas aún, como* X *es una variable producida en* G, $X \notin var(\sigma)$ *por la condición de admisibilidad* **SL**, *y entonces* $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $Y \mapsto s \in \sigma$. *Finalmente,* $\mathcal{T} \equiv \mathbf{DC}\ (h\overline{e_m\hat{\theta}'} \to h\overline{X_m\hat{\theta}}) \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$ *es un árbol de prueba para* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\overline{e}_m \to X)\hat{\theta}' \Leftarrow \Pi$. *Por tanto, como* $\hat{\theta} =_{\setminus\{\overline{X}_m\}} \hat{\theta}'$ *se concluye que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G)$.

EL *Asumamos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G')$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta} \Leftarrow \Pi$. *Definimos* $\hat{\theta}'(X) = \bot$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para toda* $Y \neq X$ *(observamos que* $\hat{\theta}'$ *está bien definida, puesto que* X *es una variable producida y no demandada en* G*). Se cumple que* $\hat{\theta} =_{\setminus\{X\}} \hat{\theta}'$. *Puesto que* $X \notin var(P \sqcap C \sqcap S \sqcap \sigma)$, *directamente tenemos que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $Y \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta}' \Leftarrow \Pi$. *Mas aún,* $\mathcal{T} \equiv \mathbf{TI}\ (e\hat{\theta}' \to \bot \Leftarrow \Pi, [\,])$ *es un árbol de prueba para* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (e \to X)\hat{\theta}' \Leftarrow \Pi$. *Por tanto, como* $\hat{\theta} =_{\setminus\{X\}} \hat{\theta}'$ *concluimos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G)$.

PF *Asumamos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G')$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $\Pi \models_{\mathcal{D}} (p\bar{t}_n \to! X)\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e_j \to X_j)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$ *(si* $q = 0$ *estas pruebas se omiten). Definimos* $\hat{\theta}'(X_j) = X_j$ *para cada* $1 \leq j \leq q$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para cada* $Y \notin \{X_1, \ldots, X_q\}$. *Se cumple que* $\hat{\theta}' =_{\setminus\{\overline{X}_q\}} \hat{\theta}$. *Como* $\overline{X}_q$ *son variables nuevas,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $Y \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta}' \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta}' \to X_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Puesto que* $e_j \notin Pat_{\bot}(\mathcal{U})$ *para todo* $1 \leq j \leq q$, *tenemos* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta}' \to t_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Mas aún, para cada* $e_i \in Pat_{\bot}(\mathcal{U})$ *sabemos que* $e_i \equiv t_i$ *y trivialmente* $\Pi \models_{\mathcal{D}} e_i\hat{\theta}' \sqsupseteq t_i\hat{\theta}$. *Por la* Propiedad de Aproximación *podemos obtener árboles de prueba* $\mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \to t_i\hat{\theta} \Leftarrow \Pi$. *En consecuencia, tenemos árboles de prueba* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \to t_i\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$. *Por tanto, podemos construir un árbol de prueba* $\mathcal{T} \equiv \mathbf{PF}\ (p\overline{e}_n\hat{\theta}' \to X\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$ *(*$X\hat{\theta} \neq \bot$ *pues* $X \in DVar_{\mathcal{D}}(G')$ *y* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G')$ *con* $\hat{\theta} =_{\setminus evar(G)} \theta$, *luego por el* Lema de Demanda $\hat{\theta}(X) \neq \bot$*). Finalmente, como* $\overline{X}_q$ *son nuevas variables distintas de* X, $\hat{\theta}(X) \equiv \hat{\theta}'(X)$ *y tenemos que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (p\overline{e}_n \to X)\hat{\theta}' \Leftarrow \Pi$. *Por tanto, como* $\hat{\theta} =_{\setminus\{\overline{X}_q\}} \hat{\theta}'$ *concluimos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G)$.

DT$_1$ *Asumamos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G')$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n \to X)\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional* $\mathcal{T}_{f\overline{X}_n}$ *sólo se utiliza para controlar la computación). En consecuencia, directamente se concluye que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G)$.

DT$_2$ *Asumamos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G')$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta} \Leftarrow \Pi$, $\mathcal{P} \vdash_{\mathcal{D}} (X'\ \overline{a}_k \to$

$X)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n \to X')\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional* $\mathcal{T}_{f\overline{X}_n}$ *sólo se utiliza para controlar la computación). Aplicando el* Lema de Demanda *se tiene que* $\hat{\theta}(X') \neq \bot$*, pues* X' *es una variable demandada (por las condiciones de la regla* $X \in DVar_{\mathcal{D}}(G)$*, y también* $X \in DVar_{\mathcal{D}}(G')$ *pues ni* P *ni* S *en* G *cambian, luego por definición de variable demandada* X' *también lo es en* G'*, de acuerdo con la condición de admisibilidad* **DT** *para* G'*). Por tanto, tenemos que* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}}\ ((f\overline{e}_n \to X')\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r])$ *es un árbol de prueba para* $\mathcal{T}' : \mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n \to X')\hat{\theta} \Leftarrow \Pi$*, usando* $(f\ \overline{t}_n \to r \Leftarrow P' \sqsquare C') \in [\mathcal{P}]_{\bot}$ *y siendo* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to t_i \Leftarrow \Pi\ (1 \leq i \leq n)$*,* $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P' \sqsquare C' \Leftarrow \Pi$ *y* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to \hat{\theta}(X') \Leftarrow \Pi$*. Ahora bien, como tenemos también* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(X')\ \overline{a_k\hat{\theta}} \to \hat{\theta}(X) \Leftarrow \Pi$ *y* $\hat{\theta}(X') \in Pat_{\bot}(\mathcal{U})$*, podemos construir el árbol de prueba* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ ((f\ \overline{e}_n\ \overline{a}_k \to X)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s\ ])\ (k > 0)$ *para* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n\ \overline{a}_k \to X)\hat{\theta} \Leftarrow \Pi\ (k > 0)$*. En consecuencia, se concluye que* $\Pi \sqsquare \theta \in Ans_{\mathcal{P}}(G)$*.*

FV *Asumamos que* $\Pi \sqsquare \hat{\theta} \in Ans_{\mathcal{P}}(G')$*. Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$*,* $F\hat{\theta} \equiv \overline{hX_m\hat{\theta}}$ *para* $F \mapsto h\overline{X}_m \in \sigma_0$*,* $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$*,* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsquare C)\sigma_0\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\ \overline{X}_m\ \overline{a}_k \to X)\sigma_0\hat{\theta} \Leftarrow \Pi$*. Definimos* $\hat{\theta}'(X_i) = X_i$ *para cada* $1 \leq i \leq m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para toda* $Y \notin \{\overline{X}_m\}$*. Se cumple que* $\sigma_0\hat{\theta} =_{\setminus\{\overline{X}_m\}} \hat{\theta}'$*. Como* $\overline{X}_m$ *son variables nuevas,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$*,* $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $Y \mapsto s \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsquare C)\hat{\theta}' \Leftarrow \Pi$*. Mas aún, como* $F \notin \{\overline{X}_m\}$ *y* $\hat{\theta}'(F) = \overline{hX_m\hat{\theta}}$*, tenemos que* $\mathcal{P} \vdash_{\mathcal{D}} (F\ \overline{a}_k \to X)\hat{\theta}' \Leftarrow \Pi$*. Por tanto, como* $\hat{\theta} =_{\setminus\{F,\overline{X}_m\}} \hat{\theta}'$*, concluimos que* $\Pi \sqsquare \theta \in Ans_{\mathcal{P}}(G)$*.*

CSS *Asumamos que* $\Pi \sqsquare \theta \in Ans_{\mathcal{P}}(G')$*. Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$*,* $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsquare C)\hat{\theta} \Leftarrow \Pi$*,* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional sólo se utiliza para controlar la computación). En consecuencia, directamente se tiene que* $\Pi \sqsquare \theta \in Ans_{\mathcal{P}}(G)$*.*

DI *Asumamos que* $\Pi \sqsquare \theta \in Ans_{\mathcal{P}}(G')$*. Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$*,* $Y\hat{\theta} \equiv (h_i\overline{Y}_{m_i})\hat{\theta}$ *para* $Y \mapsto h_i\overline{Y}_{m_i} \in \sigma_0$*,* $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$*,* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsquare C)\sigma_0\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\sigma_0\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional sólo se utiliza para controlar la computación). Debido a la condición de admisibilidad* **NC***,* $Y \neq R$*. Como* $Dom(\sigma_0) = \{Y\}$*,* $R\sigma_0\hat{\theta} = R\hat{\theta}$*. Mas aún, como* $e|_{pos(X,\tau)} = Y$*,* $(e|_{pos(X,\tau)})\sigma_0\hat{\theta} = (h_i\overline{Y}_{m_i})\hat{\theta} = Y\hat{\theta}$*. Asimismo, para cualquier* $Z \notin \{R, Y\}$ *se cumple que* $Z\sigma_0\hat{\theta} = Z\hat{\theta}$*. Por tanto,* $\sigma_0\hat{\theta} = \hat{\theta}$*. En consecuencia,* $\Pi \models_{\mathcal{D}} S\hat{\theta}$*,* $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$*,* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsquare C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$*. Por tanto, como* $evar(G) \subseteq evar(G')$ *y los* $\overline{Y}_{m_i}$ *son variables nuevas que no aparecen en* G*,* $\theta =_{\setminus evar(G)} \hat{\theta}$ *y se tiene finalmente que* $\Pi \sqsquare \theta \in Ans_{\mathcal{P}}(G)$*.*

DN *Asumamos que* $\Pi \sqsquare \theta \in Ans_{\mathcal{P}}(G')$*. Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que*

$\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$, $\mathcal{P} \vdash_{\mathcal{D}} (e|_{pos(X,\tau)} \to R')\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e[R']_{pos(X,\tau)} \to R\ )\hat{\theta} \Leftarrow \Pi$. *Como* $\hat{\theta} \in Sub_{\perp}(\mathcal{U})$, *también* $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta}|_{pos(X,\tau)} \to \hat{\theta}(R') \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta}[\hat{\theta}(R')]_{pos(X,\tau)} \to \hat{\theta}(R) \Leftarrow \Pi$ *son probables, donde* $\hat{\theta}(R') \in Pat_{\perp}(\mathcal{U})$. *Por la* Propiedad de Particionado, $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta} \to \hat{\theta}(R) \Leftarrow \Pi$, *es decir,* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$. *Por tanto, como* $evar(G) \subseteq evar(G')$ *y* R' *es una variable nueva,* $\hat{\theta} =_{\setminus evar(G)} \theta$ *y se tiene finalmente que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$.

RRA *Asumamos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G')$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Y \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$, $\mathcal{T}_{c_i} : \mathcal{P} \vdash_{\mathcal{D}} (P_i \ \square \ C_i)\sigma_c\hat{\theta} \Leftarrow \Pi$, $\mathcal{T}_{r_i} : \mathcal{P} \vdash_{\mathcal{D}} (r_i\sigma_c \to R)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (\sigma_f(R_j) \to R_j)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq m$. *Aplicando el* Lema de Demanda, *se tiene que* $\hat{\theta}(R) \neq \perp$ *(*R *es una variable demandada en* G *y en* G'*). Asumimos que* τ *es de la forma* $f\bar{t}_n$ *y entonces* $e = \tau\sigma = f\overline{t_n\sigma}$ *y* $e\hat{\theta} = f\overline{t_n\sigma\hat{\theta}}$. *Por tanto, una prueba de* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *usa la* $CRWL(\mathcal{D})$*-regla de deducción* $\mathbf{DF}_{\mathcal{P}}$ *con una instanciación parcial por* $\sigma_c\hat{\theta}$ *de la regla de programa* $(f\bar{t}_n \to r_i \Leftarrow P_i \ \square \ C_i) \in \mathcal{P}$ *(para algún* $1 \leq i \leq k$*) de forma que* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ (f\overline{t_n\sigma\hat{\theta}} \to \hat{\theta}(R) \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_{c_i}, \mathcal{T}_{r_i}])$ *siendo* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} (t_j\sigma \to t_j\sigma_c)\hat{\theta} \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$. *Por otra parte, para cada variable* $X \in var(t_j)$, $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(\sigma(X)) \to \hat{\theta}(\sigma_c(X)) \Leftarrow \Pi$ *es probable. En efecto:*

- *Si* $X \notin dom_f(\sigma)$, *es claro que* $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(\sigma(X)) \to \hat{\theta}(\sigma_c(X)) \Leftarrow \Pi$ *debido a que* $\sigma(X) = \sigma_c(X)$ *y a que* $\Pi \models_{\mathcal{D}} \hat{\theta}(\sigma_c(X)) \sqsupseteq \hat{\theta}(\sigma_c(X))$ *con* $\hat{\theta}(\sigma_c(X)) \in Pat_{\perp}(\mathcal{U})$.
- *Si* $X \in dom_f(\sigma) = \{R_1, \ldots, R_m\}$, $\hat{\theta}(\sigma(X)) = \hat{\theta}(\sigma_f(X))$ *y* $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(\sigma_f(X)) \to \hat{\theta}(X)\ (= \hat{\theta}(\sigma_c(X))) \Leftarrow \Pi$ *es probable en* $CRWL(\mathcal{D})$ *por hipótesis inicial.*

Como $evar(G) \subseteq evar(G')$ *y* $\overline{X}$ *son variables nuevas,* $\hat{\theta} =_{\setminus evar(G)} \theta$ *y se tiene finalmente que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$.

CS *Asumamos que* $\Pi_i \ \square \ \theta_i \in Ans_{\mathcal{P}}(G_i)$ $(1 \leq i \leq k)$. *Existe* $\hat{\theta}_i =_{\setminus evar(G_i)} \theta_i$ *tal que* $\Pi_i \models_{\mathcal{D}} S_i\hat{\theta}_i$, $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $X \mapsto t \in \sigma_i$, $Y\hat{\theta}_i \equiv s\hat{\theta}_i$ *para cada* $Y \mapsto s \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i$. *Definimos* $\hat{\theta'}_i(Y) = Y$ *para todo* $Y \in \overline{Y}_i \setminus Dom(\sigma_i)$ *y* $\hat{\theta'}_i(Z) = \hat{\theta}_i(Z)$ *para* $Z \in (\setminus\{\overline{Y}_i\}) \cup Dom(\sigma_i)$. *Como* $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $X \mapsto t \in \sigma_i$, *se cumple que* $\sigma_i\hat{\theta}_i =_{(\setminus\{\overline{Y}_i\}) \cup Dom(\sigma_i)} \hat{\theta'}_i$. *Como* $\overline{Y}_i$ *son variables nuevas,* $Y\hat{\theta'}_i \equiv Y\hat{\theta}_i \equiv s\hat{\theta}_i \equiv s\hat{\theta'}_i$ *para cada* $Y \to s \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta'}_i \Leftarrow \Pi_i$. *Ahora, probamos* $\Pi_i \models_{\mathcal{D}} S\hat{\theta'}_i$. *Sea* $\mu \in Sol_{\mathcal{D}}(\Pi_i)$. *Como* $\Pi_i \models_{\mathcal{D}} S_i\hat{\theta}_i$, *tenemos que* $\mu \in Sol_{\mathcal{D}}(S_i\hat{\theta}_i)$. *De aquí,* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Además, como* $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $X \mapsto t \in \sigma_i$ *también tenemos que* $X\hat{\theta}_i\mu \equiv t\hat{\theta}_i\mu$ *para cada* $X \mapsto t \in \sigma_i$. *Por tanto, existe* $\hat{\theta'}_i\mu =_{\setminus S} \hat{\theta}_i\mu$ *tal que* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S_i)$ *y* $X\hat{\theta}_i\mu \equiv t\hat{\theta}_i\mu$ *para cada* $X \mapsto t \in \sigma_i$. *Por definición,* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(\ \exists_{\setminus S}.\ S_i \ \square \ \sigma_i)$ *(*$1 \leq i \leq k$*) y entonces* $\hat{\theta}_i\mu \in \bigcup_{i=1}^{k} Sol_{\mathcal{D}}\ (\exists_{\setminus S}.\ S_i \ \square \ \sigma_i)$. *Usando los requerimientos de un resolutor de restricciones (ver* Definición 36 2.(c)*), se sigue que también* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S)$. *Entonces,* $\mu \in Sol_{\mathcal{D}}(S\hat{\theta}_i)$. *Por tanto, con-*

cluimos que $\Pi_i \square \theta_i \in Ans_{\mathcal{P}}(G)$ *para cada* $1 \leq i \leq k$.

AC *Asumamos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$. *Existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $\Pi \models_{\mathcal{D}} (p\bar{t}_n \to! \; t)\hat{\theta}$, $X\hat{\theta} \equiv s\hat{\theta}$ *para cada* $X \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e_j \to X_j)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$ *(si* $q = 0$ *estas pruebas se omiten). Definimos* $\hat{\theta}'(X_j) = X_j$ *para cada* $1 \leq j \leq q$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para cada* $Y \notin \{X_1, \ldots, X_q\}$. *Se cumple que* $\hat{\theta}' =_{\setminus \{\overline{X}_q\}} \hat{\theta}$. *Como* $\overline{X}_q$ *son variables nuevas,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $X\hat{\theta}' \equiv X\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $X \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta}' \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta}' \to X_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Puesto que* $e_j \notin Pat_{\perp}(\mathcal{U})$ *para todo* $1 \leq j \leq q$, *tenemos* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta}' \to t_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Mas aún, para cada* $e_i \in Pat_{\perp}(\mathcal{U})$ *sabemos que* $e_i \equiv t_i$ *y trivialmente* $\Pi \models_{\mathcal{D}} e_i\hat{\theta}' \sqsupseteq t_i\hat{\theta}$. *Por la* Propiedad de Aproximación *podemos obtener árboles de prueba* $\mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \to t_i\hat{\theta} \Leftarrow \Pi$. *Por tanto, tenemos árboles de prueba* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \to t_i\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$. *Como* $\Pi \models_{\mathcal{D}} (p\bar{t}_n \to! \; t)\hat{\theta}$, *podemos construir un árbol de prueba* $\mathcal{T} \equiv \mathbf{AC}\ (p\bar{e}_n\hat{\theta}' \to t\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$ *(*$t\hat{\theta} \neq \perp$ *pues* $t \notin \mathcal{V}$ *o* $t \in DVar_{\mathcal{D}}(G')$ *y aplicaríamos el* Lema de Demanda). *Finalmente, como* $\overline{X}_q$ *son variables nuevas,* $t\hat{\theta} \equiv t\hat{\theta}'$ *y tenemos que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (p\bar{e}_n \to! \; t)\hat{\theta}' \Leftarrow \Pi$. *Por tanto, como* $\hat{\theta} =_{\setminus \{\overline{X}_q\}} \hat{\theta}'$ *concluimos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$.

□

El siguiente resultado de corrección se deduce facilmente a partir del *Lema de Corrección* y asegura que las respuestas computadas para un objetivo G son realmente respuestas correctas de G.

Teorema 67 (**Corrección de** $CDNC(\mathcal{D})$)**.** *Si* G_0 *es un objetivo inicial y* $G_0 \vdash^*_{CDNC(\mathcal{D})} G_n$,*donde* $G_n \equiv \exists\overline{U}.\ \square\, \square\, S \square \sigma$ *es un objetivo resuelto, entonces* $S \square \sigma \in Ans_{\mathcal{P}}(G_0)$.

Demostración. *Como* $S \square \sigma \in Ans_{\mathcal{P}}(G_n)$, *si aplicamos varias veces hacia atras el item 3 del* Lema de Corrección, *obtenemos* $S \square \sigma \in Ans_{\mathcal{P}}(G_0)$.

□

La completitud de $CDNC(\mathcal{D})$ está basada en la siguiente idea: siempre que $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$ y G no está resuelto todavía, hay un número finito de opciones para un primer paso de cómputo $G \Vdash G_j$ $(1 \leq j \leq l)$ tal que el nuevo objetivo G_j está "más cerca de ser resuelto" y "cubre todas las soluciones de $\Pi \square \theta$". Esta idea se precisa en el siguiente lema, el cual depende de un *orden de progreso bien fundado* $\rhd$ para objetivos admisibles, que combinan técnicas parecidas a las usadas en [22] para demostrar completitud del cálculo de estrechamiento perezoso para FLP-lenguajes y [51] para $CFLP$-lenguajes.

Lema 68 (**Lema de Progreso).** *Asumamos un objetivo admisible G en forma no resuelta, y una respuesta testigo no trivial $\mathcal{M} : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$. Entonces:*

1. *Hay alguna $CDNC(\mathcal{D})$-regla de transformación aplicable a G.*
2. *Para cualquier $CDNC(\mathcal{D})$-regla R aplicable a G, existe l objetivos G_j con respuestas testigos no triviales $\mathcal{M}_j : \Pi_j \,\square\, \theta_j \in Ans_{\mathcal{P}}(G_j)$ $(1 \leq j \leq l)$ tales que:*
   - $G \vdash_R G_j$ *para cada* $1 \leq j \leq l$,
   - $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) \subseteq \bigcup_{j=1}^{l} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_j \,\square\, \theta_j)$,
   - $(G, \Pi, \theta, \mathcal{M}) \rhd (G_j, \Pi_j, \theta_j, \mathcal{M}_j)$ *para cada* $1 \leq j \leq l$, *donde* $\rhd$ *es una combinación de los ordenes de progreso bien fundados definidos en la Figura A.1.*

Demostración. *(1) Si $G \equiv \exists \overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma$ es un objetivo admisible en forma no resuelta, entonces P o C es no vacío. Procederemos asumiendo de forma gradual que ninguna regla, excepto una (llamada* **EL***) debe ser aplicable a G, y entonces concluiremos que esta regla sobrante es* **EL***. Obsevamos que las reglas de fallo no pueden ser aplicables puesto que en cualquier otro caso G no tendría solución, debido al item 2 del* Lema de Corrección*. Asumamos que* **AC** *no es aplicable. Entonces, C debe ser vacío y el objetivo tiene la forma $G \equiv \exists \overline{U}.\ P \,\square\, \square\, S \,\square\, \sigma$ con P no vacío. Ahora asumamos que* **SS**, **IM**, **PF**, **DT**, **FV** *no son aplicables en suspensiones y* **CSS**, **DI**, **DN**, **RRA** *no son aplicables en producciones demandadas. Entonces debe ser el caso en el que todas las producciones en P son de una de las dos formas siguientes:*

1. *$h\overline{e}_m \to R$ o $f\overline{e}_n\overline{a}_k \to R$ $(k \geq 0)$ or $p\overline{e}_n \to R$ o $F\overline{a}_k \to R$ $(k > 0)$, donde $h\overline{e}_m$ es una expresión rígida no pasiva (pero no un patrón) y en todos los casos R es una variable producida pero no demandada, en particular $R \notin DVar_{\mathcal{D}}(S)$.*
2. *$R'\overline{a}_k \to R$ $(k > 0)$ o $< e, \underline{case}(\tau,\ X,\ [\mathcal{T}_1,\ \ldots,\ \mathcal{T}_k]) > \to R$ con $e|_{pos(\tau,X)} = R'$, y ambos R y R' son variables producidas y demandadas (debido a la condición de admisibilidad* **DT** *y* Definición 48*).*

Sin embargo, no pueden aparecer producciones en P de la forma (2) (en cualquier otro caso, podríamos encontrar una producción de la forma (2) y otra suspensión $e \to R'$ de la forma (1) con R' una variable producida y demandada (contradicción). Así pues, todas las producciones de P deben ser de la forma (1) mencionada anteriormente. Consideremos el conjunto χ de tales R's, que es, $\chi =_{def} pvar(G)$. Si la regla **CS** *no es aplicable, S debe estar en χ-forma resuelta. Pero entonces, debido al hecho de que $\chi \cap DVar_{\mathcal{D}}(S) = \emptyset$ y al requisito (a) de los resolutores de restricciones de la* Definición 36*, concluimos $\chi \cap var(S) = \emptyset$. Elijamos ahora algún*

$R \in \chi$ minimal en la relación $\gg_P^+$ (puesto que el minimal de los elementos existe, debido al número finito de variables que aparecen en G y a la propiedad **NC** *de objetivos admisibles). Tal R no puede aparecer en ninguna otra producción en P ni en la sustitución σ del objetivo por la condición de admisibilidad* **SL** *y entonces se verifica que $R \notin var(P \sqcap C \sqcap S \sqcap \sigma)$. Por tanto, la regla* **EL** *puede ser aplicada a la producción donde aparece R.*

(2) En cada uno de los siguientes casos, G' y G_i son los objetivos obtenidos por la aplicación de la correspondiente $CDNC(\mathcal{D})$-transformación.

SS *Sea $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G)$. Entonces existe $\hat{\theta} =_{\setminus evar(G)} \theta$ tal que $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ para cada $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta} \Leftarrow \Pi$ y $\mathcal{P} \vdash_{\mathcal{D}} (t \to X)\hat{\theta} \Leftarrow \Pi$. Debido a la* Propiedad de Aproximación*, $\Pi \models_{\mathcal{D}} t\hat{\theta} \sqsupseteq X\hat{\theta}$. Si consideramos $\hat{\theta}' = \sigma_0\hat{\theta}$, se cumple que $\hat{\theta}' =_{\setminus\{X\}} \hat{\theta}$, $\hat{\theta}' \sqsupseteq \hat{\theta}$ y $\sigma_0\hat{\theta}' = \hat{\theta}'$. Es decir, es posible elevar $\hat{\theta}$ para obtener $\hat{\theta}' \sqsupseteq \hat{\theta}$ tal que $\hat{\theta}' =_{\setminus\{X\}} \hat{\theta}$ y $\sigma_0\hat{\theta}' \sqsupseteq \hat{\theta}$. Se sigue que $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$ (si X aparece en S, $X\sigma_0\hat{\theta}' \equiv t\hat{\theta}' \equiv t\hat{\theta}$ pues $X \notin var(t)$ y sabemos que $\sigma_0\hat{\theta}' \sqsupseteq \hat{\theta}$), $Z\hat{\theta}' \equiv s\hat{\theta}'$ para cada $Z \mapsto s \in \sigma$ (la variable X no puede aparecer en σ pues se trata de una variable producida) y $((P \sqcap C)\hat{\theta} \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \sqcap C)\sigma_0\hat{\theta}' \Leftarrow \Pi)$. Usando la* Propiedad de Encubrimiento*, también tenemos que $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\sigma_0\hat{\theta}' \Leftarrow \Pi$. Así pues, si tomamos $\hat{\theta}' =_{\setminus evar(G')} \theta'$ entonces $\Pi \sqcap \theta' \in Ans_{\mathcal{P}}(G')$. Claramente, $Sol_{\mathcal{D}}(\Pi \sqcap \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \sqcap \theta')$.*

IM *Sea $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G)$. Entonces existe $\hat{\theta} =_{\setminus evar(G)} \theta$ tal que $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ para cada $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta} \Leftarrow \Pi$ y $\mathcal{P} \vdash_{\mathcal{D}} (h\overline{e}_m \to X)\hat{\theta} \Leftarrow \Pi$. Como $X \in DVar_{\mathcal{D}}(P \sqcap S)$, aplicando el* Lema de Demanda *se tiene que $\hat{\theta}(X) \neq \bot$. Además, como por hipótesis $\Pi \sqcap \theta$ es una respuesta no trivial para G, se tiene que $Sat_{\mathcal{D}}(\Pi)$. Por tanto, un árbol de prueba $\mathcal{T}$ para $\mathcal{P} \vdash_{\mathcal{D}} (h\overline{e}_m \to X)\hat{\theta} \Leftarrow \Pi$ tendrá la forma $\mathcal{T} \equiv \mathbf{R}\ ((h\overline{e}_m \to X)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$ con $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to t_i \Leftarrow \Pi$ para cada $1 \leq i \leq m$. Si $\hat{\theta}(X)$ es de la forma $h\overline{t}_m$ entonces* **R** *es la regla* **DC** *y si $\hat{\theta}(X)$ es una variable, entonces $\Pi \models_{\mathcal{D}} h\overline{t}_m \sqsupseteq \hat{\theta}(X)$ y* **R** *es la regla* **IR**. *Definimos $\hat{\theta}'(X_i) = t_i$ para todo $1 \leq i \leq m$ y $\hat{\theta}'(Y) = \hat{\theta}(Y)$ para todo $Y \notin \{\overline{X}_m\}$. Se cumple que $\sigma_0\hat{\theta}' =_{\setminus\{\overline{X}_m\}} \hat{\theta}$ si* **R** *es* **DC** *y $\Pi \models_{\mathcal{D}} \sigma_0\hat{\theta}' \sqsupseteq_{\setminus\{\overline{X}_m\}} \hat{\theta}$ si* **R** *es* **IR**. *En ambos casos, como $\overline{X}_m$ son variables nuevas y X es una variable producida, ésto implica que $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$, $Z\hat{\theta}' \equiv s\hat{\theta}'$ para cada $Z \mapsto s \in \sigma$ y $(e_i\hat{\theta} \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e_i\sigma_0\hat{\theta}' \to t_i \Leftarrow \Pi)$ para cada $1 \leq i \leq m$. Usando la* Propiedad de Encubrimiento*, $\mathcal{T}'_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\sigma_0\hat{\theta}' \to t_i \Leftarrow \Pi$ $(1 \leq i \leq m)$. Como $\hat{\theta}'(\sigma_0(X_i)) = t_i$, tenemos que $\mathcal{T}'_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \to X_i)\sigma_0\hat{\theta}' \Leftarrow \Pi$ para cada $1 \leq i \leq m$. Mas aún, también tenemos que $\overline{\mathcal{T}'} : \mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\sigma_0\hat{\theta}' \Leftarrow \Pi$ aplicando de nuevo la* Propiedad de Encubrimiento*. De aquí, si tomamos $\theta' =_{\setminus evar(G')} \hat{\theta}'$ entonces $\Pi \sqcap \theta' \in Ans_{\mathcal{P}}(G')$. Claramente, $Sol_{\mathcal{D}}(\Pi \sqcap \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \sqcap \theta')$.*

EL *Sea* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to X)\hat{\theta} \Leftarrow \Pi$. *Por tanto, directamente se tiene que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$ *y que* $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \square \theta)$ *debido a que* $X \notin var(P \square C \square S \square \sigma)$.

PF *Sea* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (p\overline{e}_n \to X)\hat{\theta} \Leftarrow \Pi$. *Como* $X \in DVar_{\mathcal{D}}(P \square S)$, *por el* Lema de Demanda *sabemos que* $\hat{\theta}(X) \neq \bot$. *Además, se tiene que* $Sat_{\mathcal{D}}(\Pi)$ *pues* $\Pi \square \theta$ *es una respuesta no trivial. Por tanto,* $\mathcal{T} \equiv \mathbf{PF}\ ((p\overline{e}_n \to X)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$ *con* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to t'_i \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$ *y* $\Pi \models_{\mathcal{D}} p\overline{t'}_n \to! \hat{\theta}(X)$. *Para cada* $1 \leq i \leq n$, *si* $e_i \in Pat_{\bot}(\mathcal{U})$ *entonces* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq e_i\hat{\theta}$ *por la* Propiedad de Aproximación. *Definimos* $\hat{\theta}'(X_j) = t'_j$ *para cada* $1 \leq j \leq q$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para toda* $Y \notin \{\overline{X}_q\}$ *(si* $q = 0$ *entonces tomamos* $\hat{\theta}' = \hat{\theta}$*). Tenemos árboles de prueba* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta} \to t'_j \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Como* $\overline{X}_q$ *son variables nuevas, también tenemos que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Z\hat{\theta}' \equiv s\hat{\theta}'$ *para cada* $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta}' \Leftarrow \Pi$ *y* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} (e_j \to X_j)\hat{\theta}' \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Por otra parte, para cada* $1 \leq i \leq n$, *si* $e_i \in Pat_{\bot}(\mathcal{U})$ *entonces sabemos que* $t_i \equiv e_i$ *y* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq e_i\hat{\theta}$. *De aquí también* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq t_i\hat{\theta}'$. *En cualquier otro caso,* $t'_j = \hat{\theta}'(X_j)$, $t_j \equiv X_j$ *y tenemos que* $t'_j = t_j\hat{\theta}'$. *Por tanto, como* $\hat{\theta}(X) = \hat{\theta}'(X)$ *y puesto que* $\Pi \models_{\mathcal{D}} p\overline{t'}_n \to! \hat{\theta}(X)$, *también tenemos que* $\Pi \models_{\mathcal{D}} (p\overline{t}_n \to! X)\hat{\theta}'$. *Finalmente, si tomamos* $\theta' =_{\backslash evar(G')} \hat{\theta}'$ *entonces* $\Pi \square \theta' \in Ans_{\mathcal{P}}(G')$. *Claramente,* $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \square \theta')$.

DT$_1$ *Sea* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (f\overline{e}_n \to X)\hat{\theta} \Leftarrow \Pi$. *Puesto que los árboles definicionales son sólo usados para controlar la computación, directamente se tiene que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$ *y que* $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \square \theta)$.

DT$_2$ *Sea* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv t\hat{\theta}$ *para cada* $Z \mapsto t \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n\ \overline{a}_k \to X)\hat{\theta} \Leftarrow \Pi$. *Como* $X \in DVar_{\mathcal{D}}(P \square S)$, *el* Lema de Demanda *asegura que* $\hat{\theta}(X) \neq \bot$. *Además, puesto que* $\Pi \square \theta$ *es una respuesta no trivial de* G, *se tiene* $Sat_{\mathcal{D}}(\Pi)$. *Por tanto,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ (f\ \overline{e_n\hat{\theta}}\ \overline{a_k\hat{\theta}} \to \hat{\theta}(X) \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s])$ *usando* $(f\ \overline{t}_n \to r \Leftarrow P \square C) \in [\mathcal{P}]_{\bot}$ *y siendo* $s \in Pat_{\bot}(\mathcal{U})$, *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to t_i \Leftarrow \Pi$ *(*$1 \leq i \leq n$*),* $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P \square C \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} \overline{sa_k\hat{\theta}} \to \hat{\theta}(X) \Leftarrow \Pi$. *Definimos* $\hat{\theta}'(X') = s$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para toda* $Y \neq X'$. *Se tiene entonces que* $\hat{\theta}' =_{\backslash \{X'\}} \hat{\theta}$ *y puesto que* X' *es una variable nueva que no aparece en* G, *directamente se tiene que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Z\hat{\theta}' \equiv Z\hat{\theta} \equiv t\hat{\theta} \equiv t\hat{\theta}'$ *para cada* $Z \mapsto t \in \sigma$

y $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta}' \Leftarrow \Pi$. *Además, a partir de la deducción correspondiente a* $\mathcal{T}_s$ *es posible construir también la derivación* $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}'(X')\ \overline{a_k\hat{\theta}'} \to \hat{\theta}'(X) \Leftarrow \Pi$, *es decir,* $\mathcal{P} \vdash_{\mathcal{D}} (X'\ \overline{a}_k \to X)\hat{\theta}' \Leftarrow \Pi$. *Asimismo, tenemos que* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \to t_i \Leftarrow \Pi$ *(*$1 \leq i \leq n$*),* $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P \sqsubset C \Leftarrow \Pi$ *y* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to \hat{\theta}'(X') \Leftarrow \Pi$. *Puesto que se tiene* $Sat_{\mathcal{D}}(\Pi)$ *y* $\hat{\theta}'(X') = s \neq \bot$ *(de otro modo no sería deducible* $\mathcal{T}_s$ *para* $k > 0$*) podemos construir* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}}\ (f\ \overline{e_n\hat{\theta}'} \to \hat{\theta}'(X') \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r])$ *usando* $(f\ \bar{t}_n \to r \Leftarrow P \sqsubset C) \in [\mathcal{P}]_{\bot}$. *Por tanto, tenemos que* $\mathcal{T}' : \mathcal{P} \vdash_{\mathcal{D}} (f\ \bar{e}_n \to X')\hat{\theta}' \Leftarrow \Pi$. *Finalmente, si tomamos* $\theta' =_{\backslash evar(G')} \hat{\theta}'$ *entonces* $\Pi \sqsubset \theta' \in Ans_{\mathcal{P}}(G')$ *y* $Sol_{\mathcal{D}}(\Pi \sqsubset \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \sqsubset \theta')$.

FV *Sea* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$, $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (F\overline{a}_k \to X)\hat{\theta} \Leftarrow \Pi$. *Puesto que* $X \in DVar_{\mathcal{D}}(P \sqsubset S)$, *aplicando el* Lema de Demanda *se tiene que* $\hat{\theta}(X) \neq \bot$. *Mas aún,* F *también es una variable demandada y* $\hat{\theta}(F) \neq \bot$. *Como* $k > 0$, *sabemos que* $\hat{\theta}(F) \notin \mathcal{U} \cup \mathcal{V}$ *(en caso contrario, el árbol de prueba* $\mathcal{T}$ *no sería posible en el cálculo* $CRWL(\mathcal{D})$*). Por tanto,* $\hat{\theta}(F)$ *ha de ser de la forma* $h\bar{t}_m \in Pat_{\bot}(\mathcal{U})$. *Definimos* $\hat{\theta}'(X_i) = t_i$ *para cada* $1 \leq i \leq m$, $\hat{\theta}'(F) = h\bar{t}_m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{\overline{X}_m, F\}$. *Se cumple que* $\sigma_0\hat{\theta}' = \hat{\theta}'$ *y que* $\sigma_0\hat{\theta}' =_{\backslash\{\overline{X}_m\}} \hat{\theta}$. *Como* $\overline{X}_m$ *son variables nuevas, ésto implica que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$, $Z\hat{\theta}' \equiv s\hat{\theta}'$ *para cada* $Z \mapsto s \in \sigma$, $\hat{\theta}'(F) \equiv h\overline{X_m\hat{\theta}'}$ *para* $F \mapsto h\overline{X}_m \in \sigma_0$ *y* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\sigma_0\hat{\theta}' \Leftarrow \Pi$. *Mas aún, a partir de* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (F\ \overline{a}_k \to X)\hat{\theta} \Leftarrow \Pi$ *también tenemos que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\ \overline{X}_m\ \overline{a}_k \to X)\sigma_0\hat{\theta}' \Leftarrow \Pi$. *Finalmente, podemos tomar* $\theta' =_{\backslash evar(G')} \hat{\theta}'$ *y de esta forma* $\Pi \sqsubset \theta' \in Ans_{\mathcal{P}}(G')$. *Claramente,* $Sol_{\mathcal{D}}(\Pi \sqsubset \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \sqsubset \theta')$.

CSS *Sea* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional sólo se utiliza para controlar la computación). Por tanto, directamente se tiene que* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G')$ *y que* $Sol_{\mathcal{D}}(\Pi \sqsubset \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \sqsubset \theta)$.

DI *Sea* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional sólo se utiliza para controlar la computación). De acuerdo con el* Lema de Demanda, $\hat{\theta}(R) \neq \bot$ *(*$R \in DVar_{\mathcal{D}}(P \sqsubset C)$ *por las condiciones de admisibilidad de* G*). Por la estructura del árbol definicional* case $(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$ *y puesto que* $e|_{pos(X,\tau)} = Y$, *se sigue que* $\hat{\theta}(Y) = h_i\bar{t}_{m_i} \in Pat_{\bot}(\mathcal{U})$, *donde* h_i *(*$1 \leq i \leq k$*) es un símbolo pasivo con aridad* m_i *y* $t_i \in Pat_{\bot}(\mathcal{U})$. *Como además* $\Pi \sqsubset \theta$ *es una respuesta no trivial de* G, *se tiene que* $Sat_{\mathcal{D}}(\Pi)$. *Por tanto, la* $CRWL(\mathcal{D})$*-deducción* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *usa en el último paso la regla* $DF_{\mathcal{P}}$ *con una instanciación parcial de una regla de programa en* $\mathcal{P}$ *que ha de ser de la forma* $h_i\bar{s}_{m_i}$ *en la posición* $pos(X, \tau)$. *De aquí, podemos definir* $\hat{\theta}'(Y_j) = t_j$ *para cada* $1 \leq j \leq m_i$ *y* $\hat{\theta}'(X)$

$= \hat{\theta}(X)$ *para cualquier* $X \notin \{\overline{Y}_{m_i}\}$. *Esto implica que* $\sigma_0\hat{\theta}' =_{\setminus\{\overline{Y}_{m_i}\}} \hat{\theta}$. *Como las variables* $\overline{Y}_{m_i}$ *son nuevas, se tiene que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\sigma_0\hat{\theta}' \Leftarrow \Pi$ *y* $Z\sigma_0\hat{\theta}' \equiv s\sigma_0\hat{\theta}'$ *para cada* $Z \mapsto s \in \sigma$. *Mas aún, puesto que* $Y \notin \{\overline{Y}_{m_i}\}$, *se tiene que* $\hat{\theta}'(Y) = \hat{\theta}(Y) = h_i\bar{t}_{m_i}$ *y que* $(h\overline{Y}_{m_i})\hat{\theta}' = h_i\overline{Y_{m_i}\hat{\theta}'} = h_i\bar{t}_{m_i}$. *Se sigue entonces que* $Y\hat{\theta}' = (h\overline{Y}_{m_i})\hat{\theta}'$ *para* $Y \mapsto h\overline{Y}_{m_i} \in \sigma_0$. *Puesto que* $e|_{pos(X,\tau)} = Y$ *y* $\hat{\theta}'(\sigma_0(Y)) = \hat{\theta}(Y) = h_i\bar{t}_{m_i}$, $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\sigma_0\hat{\theta}' \Leftarrow \Pi$ *es deducible con la misma deducción que* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$. *Por tanto, si tomamos* $\theta' =_{\setminus evar(G')} \hat{\theta}'$ *se tiene que* $\Pi \sqsubset \theta' \in Ans_{\mathcal{P}}(G')$ *y claramente* $Sol_{\mathcal{D}}(\Pi \sqsubset \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \sqsubset \theta')$.

DN *Sea* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional sólo se utiliza para controlar la computación). De acuerdo con el* Lema de Demanda, $\hat{\theta}(R) \neq \bot$ *(*$R \in DVar_{\mathcal{D}}(P \sqsubset C)$ *por las condiciones de admisibilidad de G). Puesto que* $pos(X,\tau) \in Pos(e)$, *también* $pos(X,\tau) \in Pos(e\hat{\theta})$. *Adicionalmente, se tienen las siguientes propiedades:*

- *Tanto* $e\hat{\theta}$ *como* $e\hat{\theta}|_{pos(X,\tau)}$ *tienen el mismo símbolo de función en su raíz que* e *y* $e|_{pos(X,\tau)}$, *respectivamente.*

- *Como* $\tau \preceq e$ *y* $\tau \preceq \tau\hat{\theta}$, *también* $\tau\hat{\theta} \preceq e\hat{\theta}$ *y* $\tau \preceq e\hat{\theta}$ *con* τ *en la raíz del árbol definicional* $\underline{case}$ $(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$.

Aplicando la Propiedad de Particionado, $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta}|_{pos(X,\tau)} \to s \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta}[s]_{pos(X,\tau)} \to \hat{\theta}(R) \Leftarrow \Pi$ *para algún* $s \in Pat_{\bot}(\mathcal{U})$ *(*$s \neq \bot$*). De aquí, podemos definir* $\hat{\theta}'(R') = s$ *y* $\hat{\theta}'(X) = \hat{\theta}(X)$ *para cualquier* $X \neq R'$. *Puesto que* R' *es una variable nueva, tenemos pruebas para* $\mathcal{P} \vdash_{\mathcal{D}} (e|_{pos(X,\tau)} \to R')\hat{\theta}' \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e[R']_{pos(X,\tau)} \to R)\hat{\theta}' \Leftarrow \Pi$. *Además,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Z\hat{\theta}' \equiv s\hat{\theta}'$ *para cada* $Z \mapsto s \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta}' \Leftarrow \Pi$. *Por tanto, si tomamos* $\theta' =_{\setminus evar(G')} \hat{\theta}'$ *se tiene que* $\Pi \sqsubset \theta' \in Ans_{\mathcal{P}}(G')$ *y que* $Sol_{\mathcal{D}}(\Pi \sqsubset \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \sqsubset \theta')$.

RRA *Sea* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $Z \mapsto s \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional sólo se utiliza para controlar la computación). De acuerdo con el* Lema de Demanda, $\hat{\theta}(R) \neq \bot$ *(*$R \in DVar_{\mathcal{D}}(P \sqsubset C)$ *por las condiciones de admisibilidad de G). Podemos asumir que* τ *es de la forma* $f\bar{t}_n$ *y entonces* $e = \tau\sigma_0 = f\overline{t_n\sigma_0}$ *y* $e\hat{\theta} = f\overline{t_n\sigma_0\hat{\theta}}$. *Puesto que además* $\Pi \sqsubset \theta$ *es una respuesta no trivial, se tiene que* $Sat_{\mathcal{D}}(\Pi)$. *En consecuencia,* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *usa la* $CRWL(\mathcal{D})$*-regla* $_{\mathcal{P}}$ *con una instanciación parcial* $(\tau \to r_i \Leftarrow P_i \sqsubset C_i)\mu$ *de una regla de programa* $(\tau \to r_i \Leftarrow P_i \sqsubset C_i) \in \mathcal{P}$ *(para algún* $1 \leq i \leq k$*), donde* $\mu \in Sub_{\bot}(\mathcal{U})$ *con* $Dom(\mu) \subseteq var(\tau \to r_i \Leftarrow P_i \sqsubset C_i)$. *Por tanto, el árbol de prueba correspondiente a la deducción* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *será de la forma,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ ((e \to R)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n,$

$\mathcal{T}_c, \mathcal{T}_r]$) *donde* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} t_j\sigma_0\hat{\theta} \to t_j\mu \Leftarrow \Pi$ *(*$1 \leq j \leq n$*),* $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} (P_i \sqcap C_i)\mu \Leftarrow \Pi$ *y* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r_i\mu \to \hat{\theta}(R) \Leftarrow \Pi$*. De aquí podemos definir* $\hat{\theta}'$ *como sigue:*

- *Para toda variable* $X \in var(\tau \to r_i \Leftarrow P_i \sqcap C_i) \setminus dom_c(\sigma_0)$*,* $\hat{\theta}'(X) =_{def} \mu(X)$*. En particular,* $\hat{\theta}'(R_j) = \mu(R_j)$ *para cada* $R_j \in dom_f(\sigma_0)$*.*
- *Para toda variable* $Y \in dom_c(\sigma_0)$*,* $\hat{\theta}'(Y) =_{def} \theta(\hat{\sigma_0(Y)})$*.*
- *Para toda variable* $Z \notin var(\tau \to r_i \Leftarrow P_i \sqcap C_i)$*,* $\hat{\theta}' =_{def} \hat{\theta}(Z)$*.*

Es claro que $\hat{\theta} =_{\setminus evar(G')} \hat{\theta}'$*. Mas aún:*

- *Para cada* $R_j \in dom_f(\sigma_0)$ *(*$1 \leq j \leq m$*),* $\mathcal{T}'_j : \mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}'(\sigma_f(R_j)) \to \hat{\theta}'(R_j)$ $(= \mu(R_j)) \Leftarrow \Pi$ *es deducible con una prueba extraida de alguna deducción* $\mathcal{T}_j$ $: \mathcal{P} \vdash_{\mathcal{D}} t_j\sigma_0\hat{\theta} \to t_j\mu \Leftarrow \Pi$ *(*$1 \leq j \leq n$*) en* $\mathcal{T}$*.*
- *De acuerdo con las propiedades de* $CRWL(\mathcal{D})$ *y puesto que* $\Pi \models_{\mathcal{D}} r_i\sigma_c\hat{\theta}' \sqsupseteq r_i\mu$*,* $\hat{\theta}'(R) = \hat{\theta}(R)$ *y* $\mathcal{T}$ *tiene una deducción* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r_i\mu \to \hat{\theta}(R) \Leftarrow \Pi$*, se sigue que* $\mathcal{P} \vdash_{\mathcal{D}} r_i\sigma_c\hat{\theta}' \to \hat{\theta}'(R) \Leftarrow \Pi$ *también es deducible. Lo que queda por probar es que* $\Pi \models_{\mathcal{D}} \theta'(\hat{\sigma_c(X)}) \sqsupseteq \mu(X)$ *para cada* $X \in var(r_i)$ *(ésto implica que* $\Pi \models_{\mathcal{D}} r_i\sigma_c\hat{\theta}' \sqsupseteq r_i\mu$*):*
  - *Si* $X \notin dom_c(\sigma_0)$*, entonces* $\hat{\theta}'(\sigma_c(X)) = \hat{\theta}'(X) = \mu(X)$*, luego también* $\Pi \models_{\mathcal{D}} \hat{\theta}'(\sigma_c(X)) \sqsupseteq \mu(X)$*.*
  - *Si* $X \in dom_c(\sigma_0)$*, por las propiedades de* $CRWL(\mathcal{D})$*,* $\hat{\theta}(\sigma_c(X)) = \theta(\hat{\sigma_c(X)})$ *y* $\Pi \models_{\mathcal{D}} \hat{\theta}(\sigma_c(X)) \sqsupseteq \mu(X)$ *pues* $\hat{\theta}(\sigma_c(X))$*,* $\mu(X) \in Pat_{\perp}(\mathcal{U})$ *y* $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(\sigma_c(X)) \to \mu(X) \Leftarrow \Pi$ *tiene una derivación en* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} t_j\sigma_0\hat{\theta} \to t_j\mu \Leftarrow \Pi$ *(*$1 \leq j \leq n$*) de* $\mathcal{T}$*. Por tanto,* $\Pi \models_{\mathcal{D}} \hat{\theta}'(\sigma_c(X)) \sqsupseteq \mu(X)$*.*
- $\mathcal{P} \vdash_{\mathcal{D}} (P_i \sqcap C_i)\sigma_c\hat{\theta}' \Leftarrow \Pi$ *es también* $CRWL(\mathcal{D})$*-probable, pues* $(P_i \sqcap C_i)\sigma_c\hat{\theta}'$ $\sqsupseteq (P_i \sqcap C_i)\mu$ *y en* $\mathcal{T}_c$ *se incluyen deducciones para* $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} (P_i \sqcap C_i)\mu \Leftarrow \Pi$*.*
- *Por las condiciones de admisibilidad de* G *y la definición de* $\hat{\theta}'$*, también se tienen directamente, que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$*,* $Z\hat{\theta}' \equiv Z\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $Z \mapsto s \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta}') \Leftarrow \Pi$*.*

Por tanto, si tomamos $\theta' =_{\setminus evar(G')} \hat{\theta}'$ *se tiene que* $\Pi \sqcap \theta' \in Ans_{\mathcal{P}}(G')$ *y claramente* $Sol_{\mathcal{D}}(\Pi \sqcap \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \sqcap \theta')$*.*

CS *Sea* $\mathcal{M} : \Pi \sqcap \theta \in Ans_{\mathcal{D}}(G)$*. Existe* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$*,* $X\hat{\theta} \equiv t\hat{\theta}$ *para cada* $\{X \mapsto t\} \in \sigma$ *y* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta} \Leftarrow \Pi$*. Para cada* $1 \leq i \leq k$*, definimos* $\hat{\theta}_i = u.m.g.(\hat{\theta}, \sigma_i)$ *como el* unificador más general *de la sustitución* $\hat{\theta}$ *y* σ_i *(existe* $\rho_i, \delta_i \in Sub_{\perp}(\mathcal{U})$ *tal que* $\hat{\theta}_i = \hat{\theta}\rho_i$ *y* $\hat{\theta}_i = \sigma_i\delta_i$*) y* $\Pi_i = \Pi\rho_i \wedge S_i\delta_i$*. Tenemos:*

- $\Pi_i \models_{\mathcal{D}} S_i\hat{\theta}_i$. *Sea* $\mu \in Sol_{\mathcal{D}}(\Pi_i)$. *Por definición de* Π_i, $\mu \in Sol_{\mathcal{D}}(\Pi\rho_i)$ *y* $\mu \in Sol_{\mathcal{D}}(S_i\delta_i)$. *Por lo tanto* $\rho_i\mu \in Sol_{\mathcal{D}}(\Pi)$ *y* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Puesto que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, *tenemos* $\rho_i\mu \in Sol_{\mathcal{D}}(S\hat{\theta})$ *y* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Por lo tanto* $\hat{\theta}\rho_i\mu \in Sol_{\mathcal{D}}(S)$ *y* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Puesto que* $\hat{\theta}_i = \hat{\theta}\rho_i$, *tenemos* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S)$ *y* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$ $(1 \leq i \leq k)$. *Usando el requisito* $Sol_{\mathcal{D}}(S) = \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus S}.\ S_i \,\square\, \sigma_i)$ *de un resolutor de restricciones (ver* Definición 36*), seguimos con que también* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S_i)$ *y entonces* $\mu \in Sol_{\mathcal{D}}(S_i\hat{\theta}_i)$.

- $X\hat{\theta}_i = t\hat{\theta}_i$ *para cada* $\{X \mapsto t\} \in \sigma\sigma_i$. *Si* $\{X \mapsto t\} \in \sigma$ *entonces* $X\hat{\theta}_i \equiv X\hat{\theta}\rho_i \equiv t\hat{\theta}\rho_i \equiv t\hat{\theta}_i$. *Si* $\{X \mapsto t\} \in \sigma_i$ *entonces* $X\hat{\theta}_i \equiv X\sigma_i\delta_i \equiv t\delta_i \equiv t\sigma_i\delta_i \equiv t\hat{\theta}_i$.

- $\overline{\mathcal{T}}_i : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i$. *Tenemos* $((P \,\square\, C)\hat{\theta} \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \,\square\, C)\hat{\theta}_i \Leftarrow \Pi_i)$ *(si* $(e \to t) \in P$ *entonces existe* ρ_i *tal que* $\Pi_i \models_{\mathcal{D}} e\hat{\theta}\rho_i = e\hat{\theta}_i$, $\Pi_i \models_{\mathcal{D}} t\hat{\theta}\rho_i = t\hat{\theta}_i$ *y* $\Pi_i \models_{\mathcal{D}} \Pi\rho_i$. *Análogamente, para cada* $(p\overline{e}_n \to! \ t) \in C$*) y* $\hat{\theta}_i = \sigma_i\delta_i = (\sigma_i\sigma_i)\delta_i = \sigma_i(\sigma_i\delta_i) = \sigma_i\hat{\theta}_i$. *Entonces,* $((P \,\square\, C)\hat{\theta} \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \,\square\, C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i)$. *Puesto que* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta} \Leftarrow \Pi$, *usando la* Propiedad de Encubrimiento *también tenemos* $\overline{\mathcal{T}}_i : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i$.

Ahora, si tomamos $\theta_i =_{\setminus evar(G_i)} \hat{\theta}_i$ *para cada* $1 \leq i \leq k$, *entonces* $\mathcal{M}_i : \Pi_i \,\square\, \theta_i \in Ans_{\mathcal{D}}(G_i)$, *donde cada* $\mathcal{M}_i$ *es el resultado de sustituir los árboles de demostración* $\overline{\mathcal{T}}$ *en* $\mathcal{M}$ *por los arboles de demostración* $\overline{\mathcal{T}}_i$. *Finalmente, probamos que* $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_i \,\square\, \theta_i)$. *Sea* $\mu \in Sol_{\mathcal{D}}(\Pi \,\square\, \theta)$. *Por definición,* $\mu \in Sol_{\mathcal{D}}(\Pi)$ *y* $X\mu \equiv t\mu$ *para cada* $\{X \mapsto t\} \in \theta$. *Puesto que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, *tenemos* $\mu \in Sol_{\mathcal{D}}(S\hat{\theta})$ *y de ahí* $\hat{\theta}\mu \in Sol_{\mathcal{D}}(S)$. *Ahora, usando de nuevo el requisito* $Sol_{\mathcal{D}}(S) = \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus S}.\ S_i \,\square\, \sigma_i)$, *se deduce que* $\mu \in Sol_{\mathcal{D}}(\exists_{\setminus G}.\ S_i\hat{\theta} \,\square\, \sigma_i\hat{\theta})$ $(1 \leq i \leq k)$. *Por definición, existe* $\mu' =_{\setminus G} \mu$ *tal que* $\mu' \in Sol_{\mathcal{D}}(S_i\hat{\theta})$ *y* $X\mu' \equiv t\mu'$ *para cada* $\{X \mapsto t\} \in \sigma_i\hat{\theta}$. *Puesto que* $\hat{\theta}_i = u.m.g.(\hat{\theta}, \sigma_i)$ *obtenemos* $X\mu' \equiv t\mu'$ *para cada* $\{X \mapsto t\} \in \hat{\theta}_i$. *Además, puesto que* $\hat{\theta}_i = \sigma_i\delta_i$, $S_i\sigma_i = S_i$ *y* $\Pi_i = \Pi\rho_i \wedge S_i\delta_i$, *obtenemos* $\mu' \in Sol_{\mathcal{D}}(\Pi_i)$. *Se deduce que* $\mu \in Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_i \,\square\, \theta_i)$ $(1 \leq i \leq k)$. *Por tanto,* $\mu \in \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_i \,\square\, \theta_i)$.

AC *de forma análoga al caso* **PF**.

Finalmente, la tabla de la Figura A.2 *muestra el comportamiento de las diferentes* $CDNC(\mathcal{D})$*-transformaciones a fin de definir un* orden de progreso bien fundado $\rhd$ *para objetivos admisibles con árboles definicionales con restricciones y solapamiento.* □

Finalmente, mediante la aplicación reiterada del lema previo, el siguiente resultado de completitud es fácil de demostrar. La demostración muestra que $CDNC(\mathcal{D})$ es *fuertemente completo*, es decir, la elección local de la regla de transformación de

RULE	$\mid \mathcal{M} \mid$	$\mathcal{W}\ (G, \Pi, \theta, \mathcal{M})$	$\mathcal{D}\ (G)$	$\mid G \mid_0 =$ $(\mathcal{W}\ (G, \Pi, \theta, \mathcal{M}), \mathcal{D}(G))$	$\mid G \mid_1$	$\mid G \mid_2$	$\mid G \mid_3$	$\mid S \mid$
SS	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
IM	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$		
EL	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
PF	$\succ_{mul}$	$\succ_{mul}$		$>_{lex}$				
DT	$\succeq_{mul}$	$\succ_{mul}$		$>_{lex}$				
FV	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$>_{\mathbb{N}}$			
CSS	$\succeq_{mul}$	$\succeq_{mul}$	$\succ_{mul}$	$>_{lex}$				
DI	$\succeq_{mul}$	$\succeq_{mul}$	$\succ_{mul}$	$>_{lex}$				
DN	$\succ_{mul}$	$\succ_{mul}$		$>_{lex}$				
RRA	$\succ_{mul}$	$\succ_{mul}$		$>_{lex}$				
CS	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$
AC	$\succ_{mul}$	$\succ_{mul}$		$>_{lex}$				

Figura A.1: Orden de progreso $(G, \Pi, \theta, \mathcal{M}) \rhd (G_j, \Pi_j, \theta_j, \mathcal{M}_j)$

objetivo aplicada en cada paso puede ser una elección *don't care*.

Teorema 69 (**Completitud de** $CDNC(\mathcal{D})$)**.** *Sea G_0 un objetivo admisible inicial y $\Pi_0 \square \theta_0 \in Ans_{\mathcal{P}}(G_0)$ no trivial. Entonces existe un número finito de derivaciones terminadas en objetivos resueltos $G_0 \vdash^* G_i$ $(1 \leq i \leq k)$ tales que $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{P}}(G_i)$.*

Demostración. *Mediante la aplicación repetida del* Lema de Progreso*, podemos construir un árbol de bifurcación finito $\mathcal{T}$ con raíz $\mathcal{M}_0 : \Pi_0 \square \theta_0 \in Ans_{\mathcal{P}}(G_0)$ tal que, cada nodo $\mathcal{M} : \Pi \square \theta \in Ans_{\mathcal{P}}(G)$ asociado a un objetivo G en forma no resuelta, tiene hijos $\mathcal{M}_j : \Pi_j \square \theta_j \in Ans_{\mathcal{P}}(G_j)$ $(1 \leq j \leq l)$. Puesto que $\rhd$ es un orden bien fundado, no hay caminos infinitos en $\mathcal{T}$. Debido al Lema de* König*, $\mathcal{T}$ es un árbol finito con k hojas asociadas a los objetivos resueltos G_i $(1 \leq i \leq k)$ tal que $G_0 \vdash^* G_i$ para cada $1 \leq i \leq k$. Demostramos $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{P}}(G_i)$ por inducción en la profundidad p de $\mathcal{T}$.*

- Caso Base *($p = 0$). $\mathcal{T}$ tiene solo el nodo raíz $\mathcal{M}_0 : \Pi_0 \square \theta_0 \in Ans_{\mathcal{P}}(G_0)$, donde G_0 es un objetivo en forma resuelta. En ese caso, $k = 1$, $G_1 \equiv G_0$ y directamente $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq Sol_{\mathcal{P}}(G_0)$.*

- Caso Inductivo *($p > 0$). $\mathcal{T}$ tiene un nodo raíz $\mathcal{M}_0 : \Pi_0 \square \theta_0 \in Ans_{\mathcal{P}}(G_0)$ y subárboles $\mathcal{T}_j$ $(1 \leq j \leq l)$, cada uno de ellos con raíz $\mathcal{M}_j : \Pi_j \square \theta_j \in Ans_{\mathcal{P}}(G_j)$ y hojas asociadas a los objetivos resueltos $G_{j,i}$ $(1 \leq i \leq k_j)$. Probemos $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq \bigcup_{j=1}^{l} \bigcup_{i=1}^{k_j} Sol_{\mathcal{P}}(G_{j,i})$. Aplicando el* Lema de Progreso*, tenemos $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq \bigcup_{j=1}^{l} Sol_{\mathcal{D}}(\exists_{\setminus G_0}.\ \Pi_j \square \theta_j)$. Asumamos que $\overline{W}_j$ son las variables existencialmente cuantificadas en $\exists_{\setminus G_0}.\ \Pi_j \square \theta_j$. Por* hipótesis de inducción *para cada $1 \leq j \leq l$ (la profundidad de cada subárbol $\mathcal{T}_j$ es $p_j < p$), $Sol_{\mathcal{D}}(\Pi_j \square \theta_j) \subseteq \bigcup_{i=1}^{k_j} Sol_{\mathcal{P}}(G_{j,i})$. Además, puesto que $\overline{W}_j$ no son*

variables libres en $G_{j,i}$ *para todo* $1 \leq i \leq k_j$*, tenemos también* $Sol_{\mathcal{D}}(\exists\overline{W}_j.\ \Pi_j \square \theta_j) \subseteq \bigcup_{i=1}^{k_j} Sol_{\mathcal{P}}(G_{j,i})$*, y el resultado se sigue fácilmente de ambas inclusiones.*

□

El *Teorema de Completitud* es más fuerte y general que resultados previos relacionados para $CFLP(\mathcal{D})$-lenguajes [49].

Ahora analizaremos brevemente el comportamiento de los $CDNC(\mathcal{D})$-cómputos desde el punto de vista de la eficiencia. Puesto que los arboles definicionales en las producciones demandadas son usados para asegurar los pasos de estrechamiento necesario tal y como se ilustra en [33, 35], observamos que los cómputos en $CDNC(\mathcal{D})$ son en esencia derivaciones de estrechamiento demandado módulo a elecciones indeterministas entre solapamiento y reglas de programa con restricciones. Así pues, este mecanismo eficiente dirige (y posiblemente evita) elecciones *don't know* de reglas de programa [50], codifica derivaciones de estrechamiento necesario en el sentido de [7, 33] e interpreta $CDNC(\mathcal{D})$ adecuadamente como una especificación concreta del comportamiento computacional en los sistemas $CFLP(\mathcal{D})$.

Bibliografía

[1] http://www.open-std.org/jtc1/sc22/wg14/.

[2] http://www.sics.se/isl/sicstus.

[3] *Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 27-29 August 2003, Uppsala, Sweden.* ACM, 2003.

[4] S. Antoy. Definitional trees. In Kirchner and Levi [44], pages 143–157.

[5] S. Antoy. Optimal non-deterministic functional logic computations. In M. Hanus, J. Heering, and K. Meinke, editors, *ALP/HOA*, volume 1298 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 1997.

[6] S. Antoy. Constructor-based conditional narrowing. In *PPDP*, pages 199–206. ACM, 2001.

[7] S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. *J. ACM*, 47(4):776–822, 2000.

[8] K. R. Apt. Logic programming. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 493–574. 1990.

[9] K. R. Apt. *Principles of constraint programming.* Cambridge University Press, 2003.

[10] P. Arenas-Sánchez, A. Gil-Luezas, and F. J. López-Fraguas. Combining lazy narrowing with disequality constraints. In M. V. Hermenegildo and J. Penjam, editors, *PLILP*, volume 844 of *Lecture Notes in Computer Science*, pages 385–399. Springer, 1994.

[11] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[12] H. P. Barendregt. Functional programming and lambda calculus. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 321–363. 1990.

[13] K. L. Clark. Negation as failure. In *Logic and Data Bases*, pages 293–322, 1977.

[14] J. M. Cleva, J. Leach, and F. J. López-Fraguas. A logic programming approach to the verification of functional-logic programs. In *PPDP*, pages 9–19. ACM, 2004.

[15] A. Colmerauer and P. Roussel. The birth of prolog. In *HOPL Preprints*, pages 37–52, 1993.

[16] L. Damas and R. Milner. Principal type-schemes for functional programs. In *POPL*, pages 207–212, 1982.

[17] F. S. de Boer, M. Gabbrielli, E. Marchiori, and C. Palamidessi. Proving concurrent constraint programs correct. In *POPL*, pages 98–108, 1994.

[18] F. S. de Boer and C. Palamidessi. A fully abstract model for concurrent constraint programming. In S. Abramsky and T. S. E. Maibaum, editors, *TAPSOFT, Vol.1*, volume 493 of *Lecture Notes in Computer Science*, pages 296–319. Springer, 1991.

[19] F. S. de Boer and C. Palamidessi. On the semantics of concurrent constraint programming. In *ALPUK*, pages 145–173, 1992.

[20] F. S. de Boer and C. Palamidessi. A process algebra of concurrent constraint programming. In *JICSLP*, pages 463–477, 1992.

[21] R. del Vado Vírseda. A demand-driven narrowing calculus with overlapping definitional trees. In *PPDP* [3], pages 253–263.

[22] R. del Vado Vírseda. A demand-driven narrowing calculus with overlapping definitional trees. In *PPDP* [3], pages 253–263.

[23] R. del Vado Vírseda. Declarative constraint programming with definitional trees. In B. Gramlich, editor, *FroCos*, volume 3717 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2005.

[24] R. del Vado Vírseda. A higher-order demand-driven narrowing calculus with definitional trees. In *ICTAC*, volume 4711 of *Lecture Notes in Computer Science*, pages 169–184, 2007.

[25] R. del Vado Vírseda. A higher-order logical framework for the algorithmic debugging and verification of declarative programs. In *PPDP*, pages 49–60. ACM, 2009.

[26] R. del Vado Vírseda. Cooperation of algebraic constraint domains in higher-order functional and logic programming. In *AMAST*, volume 6486 of *Lecture Notes in Computer Science*, pages 180–200, 2011.

[27] R. del Vado Vírseda and F. Pérez Morente. A modular semantics for higher-order declarative programming with constraints. In *Proceedings of the 13th international ACM SIGPLAN symposium on Principles and practices of declarative programming*, PPDP '11, pages 41–52, New York, NY, USA, 2011. ACM.

[28] R. Echahed and W. Serwe. Defining actions in concurrent declarative programming. *Electr. Notes Theor. Comput. Sci.*, 64:176–194, 2002.

[29] M. Falaschi, M. Gabbrielli, K. Marriott, and C. Palamidessi. Compositional analysis for concurrent constraint programming. In *LICS*, pages 210–221. IEEE Computer Society, 1993.

[30] M. Gabbrielli and G. Levi. Unfolding and fixpoint semantics of concurrent constraint logic programs. In H. Kirchner and W. Wechler, editors, *ALP*, volume 463 of *Lecture Notes in Computer Science*, pages 204–216. Springer, 1990.

[31] J. F. Groote and A. Ponse. Process algebra with guards - combining hoare logic with process algebra (extended abstract). In J. C. M. Baeten and J. F. Groote, editors, *CONCUR*, volume 527 of *Lecture Notes in Computer Science*, pages 235–249. Springer, 1991.

[32] C. A. Gunter and D. S. Scott. Semantic domains. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 633–674. 1990.

[33] M. Hanus. Curry: A multi-paradigm declarative language (system description). In *WLP*, pages 0–, 1997.

[34] M. Hanus. A unified computation model for functional and logic programming. In *POPL*, pages 80–93, 1997.

[35] M. Hanus and C. Prehofer. Higher-order narrowing with definitional trees. *J. Funct. Program.*, 9(1):33–75, 1999.

[36] M. Hanus and C. Prehofer. Higher-order narrowing with definitional trees. *J. Funct. Program.*, 9(1):33–75, 1999.

[37] P. V. Hentenryck and Y. Deville. Operational semantics of constraint logic programming over finite domains. In *PLILP*, pages 395–406, 1991.

[38] P. Hudak. Conception, evolution, and application of functional programming languages. *ACM Comput. Surv.*, 21(3):359–411, 1989.

[39] P. Hudak, S. L. P. Jones, P. Wadler, B. Boutel, J. Fairbairn, J. H. Fasel, M. M. Guzmán, K. Hammond, J. Hughes, T. Johnsson, R. B. Kieburtz, R. S. Nikhil, W. Partain, and J. Peterson. Report on the programming language haskell, a non-strict, purely functional language. *SIGPLAN Notices*, 27(5):1–, 1992.

[40] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *POPL*, pages 111–119, 1987.

[41] J. Jaffar, S. Michaylov, P. J. Stuckey, and R. H. C. Yap. The clp(r) language and system. *ACM Trans. Program. Lang. Syst.*, 14(3):339–395, 1992.

[42] G. S. G. B. James Gosling, Bill Joy. *The Java Language Specification*. Third edition edition.

[43] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. Log. Program.*, 12(3&4):335–367, 1992.

[44] H. Kirchner and G. Levi, editors. *Algebraic and Logic Programming, Third International Conference, Volterra, Italy, September 2-4, 1992, Proceedings*, volume 632 of *Lecture Notes in Computer Science*. Springer, 1992.

[45] R. A. Kowalski. Predicate logic as programming language. In *IFIP Congress*, pages 569–574, 1974.

[46] G. Levi, M. Martelli, and C. Palamidessi. Failure and success made symmetric. In *NACLP*, pages 3–22, 1990.

[47] J. W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

[48] R. Loogen, F. J. López-Fraguas, and M. Rodríguez-Artalejo. A demand driven computation strategy for lazy narrowing. In M. Bruynooghe and J. Penjam, editors, *PLILP*, volume 714 of *Lecture Notes in Computer Science*, pages 184–200. Springer, 1993.

[49] F. J. López-Fraguas. A general scheme for constraint functional logic programming. In Kirchner and Levi [44], pages 213–227.

[50] F. J. López-Fraguas, M. Rodríguez-Artalejo, and R. del Vado Vírseda. A lazy narrowing calculus for declarative constraint programming. In Moggi and Warren [60], pages 43–54.

[51] F. J. López-Fraguas, M. Rodríguez-Artalejo, and R. del Vado Vírseda. A lazy narrowing calculus for declarative constraint programming. In Moggi and Warren [60], pages 43–54.

[52] F. J. López-Fraguas, M. Rodríguez-Artalejo, and R. del Vado Vírseda. Constraint functional logic programming revisited. *Electr. Notes Theor. Comput. Sci.*, 117:5–50, 2005.

[53] F. J. López-Fraguas and J. Sánchez-Hernández. *OY*: A multiparadigm declarative system. In P. Narendran and M. Rusinowitch, editors, *RTA*, volume 1631 of *Lecture Notes in Computer Science*, pages 244–247. Springer, 1999.

[54] F. J. López-Fraguas and J. Sánchez-Hernández. A proof theoretic approach to failure in functional logic programming. *TPLP*, 4(1-2):41–74, 2004.

[55] M. J. Maher. Logic semantics for a class of committed-choice programs. In *ICLP*, pages 858–876, 1987.

[56] M. J. Maher. Equivalences of logic programs. In *Foundations of Deductive Databases and Logic Programming.*, pages 627–658. Morgan Kaufmann, 1988.

[57] A. I. Marcev. The metamathematics of algebraic systems. In *volume 66 of Studies in logic and the foundations of mathematics*, North-Holland, Amsterdam-New York-Oxford-Tokyo, 1971.

[58] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.

[59] S. E. Martín, M. T. Hortalá-González, M. Rodríguez-Artalejo, R. del Vado Vírseda, F. Sáenz-Pérez, and A. J. Fernández. On the cooperation of the constraint domains , , and in cflp. *TPLP*, 9(4):415–527, 2009.

[60] E. Moggi and D. S. Warren, editors. *Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 24-26 August 2004, Verona, Italy.* ACM, 2004.

[61] J. C. G. Moreno, M. T. Hortalá-González, and M. Rodríguez-Artalejo. Polymorphic types in functional logic programming. *Journal of Functional and Logic Programming*, 2001(1), 2001.

[62] M. J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science.* Springer, 1977.

[63] M. J. O'Donnell. Equational logic as a programming language. In R. Parikh, editor, *Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, page 255. Springer, 1985.

[64] M. J. Plasmeijer and M. C. J. D. van Eekelen. *Functional Programming and Parallel Graph Rewriting.* Addison-Wesley, 1993.

[65] C. Prehofer. Solving higher-order equations: From logic to programming, 1995.

[66] C. Reade. *Elements of functional programming.* International computer science series. Addison-Wesley, 1989.

[67] M. A. Salido and F. Barber. Exploiting the constrainedness in constraint satisfaction problems. In C. Bussler and D. Fensel, editors, *AIMSA*, volume 3192 of *Lecture Notes in Computer Science*, pages 126–136. Springer, 2004.

[68] V. A. Saraswat. A somewhat logical formulation of clp synchronisation primitives. In *ICLP/SLP*, pages 1298–1314, 1988.

[69] V. A. Saraswat. The category of constraint systems is cartesian-closed. In *LICS*, pages 341–345. IEEE Computer Society, 1992.

[70] V. A. Saraswat and M. C. Rinard. Concurrent constraint programming. In *POPL*, pages 232–245, 1990.

[71] V. A. Saraswat, M. C. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *POPL*, pages 333–352, 1991.

[72] E. Shapiro. A subset of concurrent prolog and its interpreter. In *Technical Report TR-003*, Tokyo, 1983. Institute for New Generation Computer Technology (ICOT).

[73] E. Y. Shapiro and A. Takeuchi. Object oriented programming in concurrent prolog. *New Generation Comput.*, 1(1):25–48, 1983.

[74] G. Smolka. The development of oz and mozart. In P. V. Roy, editor, *MOZ*, volume 3389 of *Lecture Notes in Computer Science*, page 1. Springer, 2004.

[75] L. Sterling and E. Y. Shapiro. *The Art of Prolog - Advanced Programming Techniques, 2nd Ed.* MIT Press, 1994.

[76] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, Berkeley, 1951.

[77] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, London, 1993.

[78] M. H. van Emden. An algorithm for interpreting prolog programs. In *ICLP*, pages 56–64, 1982.

[79] H. Wu. Parallel implementation of guarded horn clauses. In B. Fronhöfer and G. Wrightson, editors, *Dagstuhl Seminar on Parallelization in Inference Systems*, volume 590 of *Lecture Notes in Computer Science*, page 369. Springer, 1990.

[80] R. Yang and H. Aiso. P-prolog: A parallel logic language based on exclusive relation. *New Generation Comput.*, 5(1):79–95, 1987.